Al Zaytona University for Science and Technology

Faculty of Engineering & Technology

Department of Control & Robotics Engineering

COOPERATION AND FORMATION OF A MULTI-ROBOT SYSTEM

Prepared By:

Raghad Hasan Daraghmah 20010001

Supervised By:

Dr. Bilal Karaki

ETCR471, Section 1

A Graduation Project submitted to the College of Engineering in partial fulfillment of the requirements for the degree of B.Sc. in Control and Robotics Engineering

ZUST

2024/2025

Dedication

To My Family,

Acknowledgement

This project is a milestone on my path to obtaining a Bachelor's degree at Al Zaytona University for Science and Technology. First, I would like to acknowledge my parents and family for their support and encouragement during this period of my academic pursuits.

To my friends, Siwar and Ghadeer thank you for being the family I had here in Palestine throughout this time of my life.

Furthermore, I would like to express my gratitude to the supervisor, Dr. Bilal Karaki, for his infinite support, excellent guidance, and trust in my abilities. He helped so much by providing valuable insights that shaped my work in this project, constructive feedback, and encouragement. I could not have done this without him.

Abstract

Multi-robot systems are crucial in modern robotics as they offer effective solutions for various applications. This project presents the design and implementation of a system comprising six robots to perform specific tasks cooperatively. Each robot is equipped with four mecanum wheels that enable precise omnidirectional movement. A localization system is proposed to determine the location of each robot in real time. The proposed localization system uses a camera, driven by AI technology, to determine the locations using the AprilTag technique. The robots exchange their locations through wireless communication to make decisions that allow them to form shapes and complete certain tasks. Simulation results are conducted to validate the system, ensuring system reliability.

المستخلص

هذا النظام يتكون من مجموعة روبوتات و الذي يعد عنصر أساسي في مجال الروبوتات الحديثة. في هذا المجال، تقدم هذه الأنظمة حلولًا فعالة لمجموعة واسعة من التطبيقات. يهدف هذا المشروع إلى تصميم وبناء نظام يتألف من ستة روبوتات مجهزة بعجلات ميكانيكية، مما يمنحها القدرة على التحرك بدقة وسلاسة في جميع الاتجاهات. من خلال التواصل اللاسلكي، يمكن لهذه الروبوتات تبادل البيانات مع بعضها البعض، مما يسمح لها باتخاذ قرارات تمكنها من التعاون مع بعضها البعض لتشكيل أشكال محددة و أداء مهام معينة. يحتوي هذا النظام على كاميرا والتي تستخدم كنظام تحديد للمواقع، مدعومة بتقنيات الذكاء الاصطناعي، للكشف عن الروبوتات وتحديد مواقعها بالوقت الفعلي عن طريق التعرف على علامات مرئية مثبتة على كل روبوت داخل المساحة الخاص بهم. كما تم استخدام أنظمة محاكاة لاختبار النظام وضمان فعاليته قبل العمل على النموذج. أخيرًا، سيكون النظام متعدد الروبوتات يعمل بشكل كامل من جميع النواحي، حيث أن جميع الروبوتات مزودة بعجلات تتيح لها الحركة و أجهزة للتواصل اللاسلكي. بينما بيئة الروبوتات مجهزة بنظام كاميرا لتحديد مواقعهم . جميع ما سبق سيسمح للروبوتات بأن تتواصل سويا وتتخذ قرارات منطقية، ومناسبة لتحقيق الهدف أو تشكيل الشكل المطلوب.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AR Augmented Reality

GPS Global Positioning System

IC Integrated Circuit

LPS Local Positioning System

MAS Multi-Agent Systems

QR Quick Response

RF Radio Frequency

RFID Radio Frequency Identification Technology

URL Uniform Resource Locator

UWB Ultra Wave Band

Chapter 1

Introduction

Contents

This project investigates the formation problem of multi-robot system. Many current man-made technologies are based on the cooperation abilities of individual agents [1]. In nature, we observe many examples of multiagent systems (MAS) in operation, such as flocks of birds flying in perfect harmony or schools of fish moving together. These natural phenomena have fascinated scholars in many fields, such as computer science, biology, and physics, leading them to investigate the principles behind such collaboration. Due to their parallel capabilities, MAS provide unusual solutions to technical problems that would be too difficult or impossible to solve alone [2].

1.1 Background and Motivation

1.1.1 Background of MAS

In the late 1980s, a number of researchers began working on mobile robots that performed coordinated tasks, giving rise to the discipline of multiagent systems [3]. Scientists' interest in MAS has grown significantly since these early studies [4][5][6]. Many real-world issues that are better modeled with a group of agents rather than a single agent are the reason for the growing interest in MAS [7].

The use of agents in robotics has been researched for more than 20 years; the first paper that described the benefits and drawbacks of agents in robotics was published in 1996 [8]. Consensus problems are fundamental to the subject of distributed computing and have a long history in computer science [9].

1.1.2 Motivation

Robots usually stand for innovation. Mostly because they represent the future and an expression of the belief that technology may enhance life in more ways than humans could ever imagine. But no matter how advanced the robot is, it is limited what can be accomplished. However, the true magic lies in communicating with and arranging for robots to achieve tasks that are difficult for them to complete individually. That is the spark for the effort behind this project.

This project holds such great value primarily because seeing the principles and concepts of mathematics and theoretical knowledge come to life is thrilling. The idea of robots collaborating is not just science fiction; it is something real and achievable. That is a problem worth solving and a future worth creating. If we can get robots to cooperate today, we can only imagine what they will be capable of tomorrow.

1.2 Problem Statement

Getting robots to collaborate effectively is not an easy task. Coordination and collaboration are essential in multi-robot systems to achieve complex goals. Whether it is mapping out an area, navigating an environment, or performing a life- saving rescue operation. Robots must exchange information, such as positions or a map of their surroundings, for the purpose of cooperating and making reasonable decisions. However, if robots behave unpredictably or their movements are poorly coordinated, things will easily go wrong. Therefore, an interesting problem arises, that is, designing a consensus protocol that guarantees cooperation among agents.

According to Cena et al. [7], there are two major issues in multi-robot systems, which are

collaboration and coordination among robots, as well as planning their movement trajectory. The authors proposed using hardware and software agents to address the identified difficulties.

1.3 Objectives

The objectives of this project are to develop a multi-robot system that achieves predefined formation patterns.

This project has the following objectives:

1. To design a coordinated system to achieve leader/leaderless formation control.
2. To design and implement a local positioning system in order to localize the robots.
3. To integrate a communication network among the robots.
4. To build a group of six small robots using Mecanum wheels.

1.4 Outcomes

The outcomes of this project included the development of a fully functional multi-robot system, where robots were equipped with wheels that enabled precise omnidirectional movement and were integrated with a centralized camera system used as a positioning framework. This system established a scalable foundation for potential future expansions to handle more complex tasks. Additionally, the project validated the concept of cooperation by demonstrating how a group of robots could coordinate through localization, exchange data in real time, and make decisions based on that data. The results contributed to the field of multi-robot systems by documenting key insights and offering

valuable information on the effective design and implementation of such system. Finally, the project delivered a working model and prototype that showcased the robots' collaborative and formation capabilities, providing a practical demonstration of their integrated communication and localization functions.

1.5 Scope of the Project

The ultimate aim behind all this effort is to overcome the gap between the theoretical foundations and practical implementation by demonstrating a smooth collaboration of multiple robots. It can be stated that this project serves as a proof of concept for the seamless cooperation of a set of robots. It examines how the concepts of mathematics and system architecture could coexist in order to offer a dynamic and efficient robotics system by demonstrating the potential of multi-robot operation.

This work is about unlocking the future of robotics cooperation, not merely about accomplishing functional goals. Manufacturing, logistics, healthcare, and disaster response are just a few of the industries that multi-robot systems have the potential to transform by employing the concepts and methods shown in the project. Envision an alliance of robots constructing roads and buildings or saving and rescuing lives in dangerous situations. A minor but significant step toward that goal is taken by implementing this project.

1.6 Significance of the Study

This project holds great value because it shows how several robots can collaborate effectively, transcending individual capabilities to actual teamwork. This study links complex concepts with practical solutions by demonstrating that coordinated robotic behavior is not only possible, but also practical. As the saying goes, "Teamwork makes the dream

work" It turns out that this is true for both humans and robots. Where this project demonstrates how robots can work together to accomplish tasks that would be far too challenging for them to complete alone. By cooperating, they demonstrate that teamwork is essential to the success of their goal.

In addition to its immediate purposes, this project improves the current understanding of robotics by demonstrating how basic sensors and AprilTags may be utilized to create major improvements in robots. It proves that creative solutions can be obtained with the right approach rather than the most cutting-edge technology. For the long run, this project in the field of MAS aims to open up new possibilities rather than only creating robots. The work being done paves the way for a future in which robot cooperation may improve our quality of life by making it easier, safer, and overall a better life.

1.7 Methodology Overview

For effective coordination and task execution, the proposed multi-robot system makes use of innovative hardware components and an external localization system. Each robot has communication module and mecanum wheels for omnidirectional mobility. Regarding localization, AprilTag technology is integrated with a centralized camera system. Given that each robot has a unique tag, the camera can precisely identify its location and orientation throughout the surroundings. The relative positions of the robots are established by processing the localization data. The robots' ability to communicate with one another ensures smooth collaboration and data sharing. Each robot uses the data to choose the best path for itself, allowing an accurate creation of desired shapes through cooperation. The outcomes of implementing all this will show how well the system creates precise shape formation.

The methodology throughout this project aims to accomplish accurate task execution and smooth coordination among multiple robots. The system guarantees flexibility and accuracy in dynamic environments by combining real-time communication, enhanced motion capabilities, and localization. The project's objective of showcasing solid and efficient multi-robot coordination is met by the robots' ability to cooperate effectively together, forming desired shapes.

1.8 Structure of the Report and Time Plan

This section summarizes the dissertation and provides a brief description of the material covered in each chapter. It also provides a timeline to illustrate the project's development.

1.8.1 Structure of The Report

- **Chapter 1: Introduction**

  Provides an overview of the project, its objectives, and the problem statement. Also, it summarizes the background and motivation of the project's topic. In addition, it shows the time plan and the expected outcomes.

- **Chapter 2: Literature Review and Preliminaries**

  Discusses previous works related to the project and presents information drawn by other researchers.

- **Chapter 3: Methodology**

  Describes the approach, tools, and techniques used in the project.

- **Chapter 4: Simulation and Analysis**

  Presents the simulation of several graphs, with an analysis of their results.

- **Chapter 5: Implementation of the Proposed System**

  This chapter explains how the project was implemented step by step.

- **Chapter 6: Experimental Results**

  This chapter focuses on the experimental setup, results of formation, and analysis.

- **Chapter 7: Conclusion and Future Work**

  Summarizes the outcomes and suggests future improvements.

1.8.2 Time Plan

First Semester (13 October 2024 – 25 December 2024)

- 13/10/2024 – Selected the project idea
- 27/10/2024 – Determined the general objectives
- October–November – Conducted literature review
- 30/11/2024 – Determined the specific objectives
- Late November – Proposed the methodology
- Late November – Designed the MAS (Multi-Agent System) and LPS (Local Positioning System)
- 20/12/2024 – Analyzed control techniques
- December – Simulated behavior in MATLAB/Simulink and analyzed consensus/-formation
- 25/12/2024 – Purchased hardware components for implementation

Second Semester (26 February 2025 – 1 June 2025)

- 26 Feb – 10 Mar: Prepared robot plates and hardware setup
- 5 Mar – 15 Mar:Assembled the first robot (mechanics and wiring)
- 16 Mar – 25 Mar: Developed and tested the motor driver software
- 26 Mar – 15 Apr: Controlled the robot and tuned PID controller and tested various trajectories
- 16 Apr – 25 Apr: Replicated the setup for the remaining five robots
- 26 Apr – 10 May: Tested individual robots for localization and communication and tuning PID
- 6 May – 15 May: Implemented different formation protocols
- 16 May – 25 May: Applied different graph topologies and tested each
- 26 May – 3 Jun: Performed final testing, recorded results, and completed documentation

Chapter 2

LITERATURE REVIEW AND PRELIMINARIES

Contents

2.1 Multi-agent Systems

The primary subject of this report is multiagent systems, which are made up of independent entities called agents where they collaborate to solve problems. Through their interactions with other agents or the environment, agents learn new contexts and behaviors. In order to accomplish the task at hand, agents then use their knowledge to make decisions and act on the environment. Due to its adaptability, MAS can be applied to problems in a variety of domains [10].

Each agent's actions impact the environment, influencing the decisions of others. Coordination control involves guiding agents to collaborate and achieve their goals [11]. Coordination presents various issues, such as consensus, connectivity, and formation, which will be discussed later.

An agent is an entity that analyzes its surroundings and uses that data to decide what to do in order to achieve its goals. The entity makes this decision and then takes the necessary action. In the definition above, four keywords can be further explained [10]. Entity describes the kind of agent. An agent can be a piece of hardware, like a thermostat, software, like security agents, or a combination of both, like a robot. Environment describes the location of the agent. The environment can be a software program when

the agent is keeping an eye on the activities of software components, a network when traffic monitoring agents are involved, etc. An agent makes decisions based on the data it senses from its surroundings. Parameters are the kind of information that an agent can detect from its surroundings. For example, a soccer robot agent's parameters include the ball's location, the opponents' and team players' positions, and their speeds. Every agent can take an action that modifies the surroundings. For instance, the position of the ball changes when a soccer robot kicks it. A series of discrete or continuous activities can be carried out by an agent. The agent can carry out an infinite number of activities in a continuous series of actions, such as a soccer match. In contrast, a discrete collection of activities, such as an agent managing a room's temperature, has a finite number of actions [10].

The purpose of each agent is to solve its task with some additional limitations, like a deadline. The agent first perceives environmental characteristics to accomplish its goal. Equipped with this information, the agent can learn more about the surroundings. An agent may also take advantage of its neighbors' knowledge. Which determines the best course of action for the agent based on the goal, past actions, and history [10].

Although an agent operating independently is capable of acting alone, agents' full potential can only be fulfilled when they cooperate with other agents. Multi-Agent Systems are groups of agents that work together to accomplish a challenging task. More flexibility, redundancy, and efficiency are typically provided by a group of agents than by a single agent. The establishment of several agents often allows for their flexible grouping to perform tasks at different locations. The robustness of the group is ensured by its numerousness, as each agent can take over in the event of a failure. With several agents rather than simply one, tasks can be finished more rapidly and accurately [1].

Agents can be arranged in architectures, featuring communication links. There are different architectures such as centralized, decentralized, and distributed coordination. The centralized architecture is made up of a group of several entities that are restricted to delivering data to the central agent, which is an agent that acts with complete control over the system's task allocation and decision making [12]. However, decentralized coordination eliminates the need for central controllers. Alternatively, each agent functions autonomously, making decisions on local data and interactions with other agents. Local autonomy is used in this design to create scalable and reliable systems [13].

In distributed cooperation, every agent in a distributed architecture is at the same level of the hierarchy and is capable of making decisions on their own without the guidance of a higher-level agent [12]. In contrast to fully decentralized systems, distributed systems imply some particular activity that includes cooperation as well as precise communication.

One of the biggest problems in multiagent systems is getting agents to cooperate without a centralized controller when each agent can only access information that is local or nearby [14]. Whose primary responsibility is to use local and surrounding information to construct a distributed protocol so that the states or agent outputs can reach a predetermined agreement [15]. To put it simply, agents of this kind do not interact with one another [16].

2.2 Consensus Protocols

This section discusses consensus in multiagent systems. To accomplish a specific task, the consensus protocol makes sure that a collection of agents maintains communication and comes to an agreement [17] and accomplish a shared global objective [18]. Consensus is the process of agreeing on a quantity of interest that is dependent on the states of

all agents in networks of agents. An interaction rule that outlines the information flow between an agent and each of its network neighbors is known as a consensus protocol or algorithm [19]. The consensus problem is solved when the agent's states arrive at a shared value of interests through local information exchange, which is a common cooperative behavior in multiagent systems [18].

Herein, we examine the presence of a leader or an agent that assigns duties and goals to the other agents according to a single global objective. MAS can be categorized as leaderless or leader-follow based on the existence or lack of such a leader [10].

Various control topologies can be used in formation control for a collection of coordinated robots, depending on the particular circumstances. The group may have one or more leaders, and other robots may follow them in a predetermined manner. Onboard sensors and computing capabilities are present in every robot. Robots may have restricted communication capabilities in some application circumstances. In general, however, not every robot has access to all of the global information. The existence of a centralized controller is typically not anticipated. Every robot must have a controller that is designed using local data. In the absence of a leader, all robots will have to cooperate by depending on a worldwide consensus to accomplish a common goal [20].

With the set of nodes $V = \{1, 2, ..., n\}$ and edges $E \subseteq V \times V$, a directed graph $G = (V, E)$ can be utilized to represent the interaction topology of a network of agents [19]. Consider a set of robots whose dynamics are individually provided as a separate integrator [21]:

$$\dot{x}_i(t) = u_i(t) \tag{2.1}$$

Where $i = 1, 2, \ldots, N$, $x_i(t) \in \mathbb{R}$ is the state of the i-th robot/agent and $u_i(t) \in \mathbb{R}$ is

the control law to be determined. The input $u_i(t)$ can use the states of the other agents if they belong to its neighbor set N_i. Any graph's agreement protocol can be expressed generally as the following:

$$u_i(t) = -\sum_{j \in N_i} [x_i(t) - x_j(t)] \tag{2.2}$$

Where N_i is the set of neighboring agents of the i-th agent, equivalently:

$$u_i(t) = -\sum_{j=1}^{N} a_{ij} [x_i(t) - x_j(t)] \tag{2.3}$$

A is the adjacency matrix, where $a_{ij} = 1$ if there is an edge between the vertex of i and j. Otherwise, $a_{ij} = 0$ [22].

2.2.1 Leaderless Consensus

Each agent in a leaderless MAS independently chooses its course of action based on its objectives. If agents work together to reach an agreement on a specific feature, then each agent's choice is influenced by the choices made by other agents [10]. Suppose that a group of robots has a single integrator for each robot's dynamics [23]:

$$\dot{x}_i(t) = u_i(t) \tag{2.4}$$

Where $i = 1, 2, \ldots, N$, $x_i(t) \in \mathbb{R}$ is the state of the i-th robot/agent and $u_i(t) \in \mathbb{R}$ is the control law to be determined. It has been claimed that the multiagent (2.4) leaderless

consensus problem is solved if [24]:

$$\lim_{t\to\infty} \|x_i(t) - x_j(t)\| = 0 \tag{2.5}$$

is satisfied for all agents $i, j \in \mathcal{V}$ and for all initial conditions.

The concept of leaderless consensus is shown in Figure 2.1. The initial conditions of the agents vary. The agents begin to follow one another. Within eight seconds, the state of each agent reaches the state of the others.

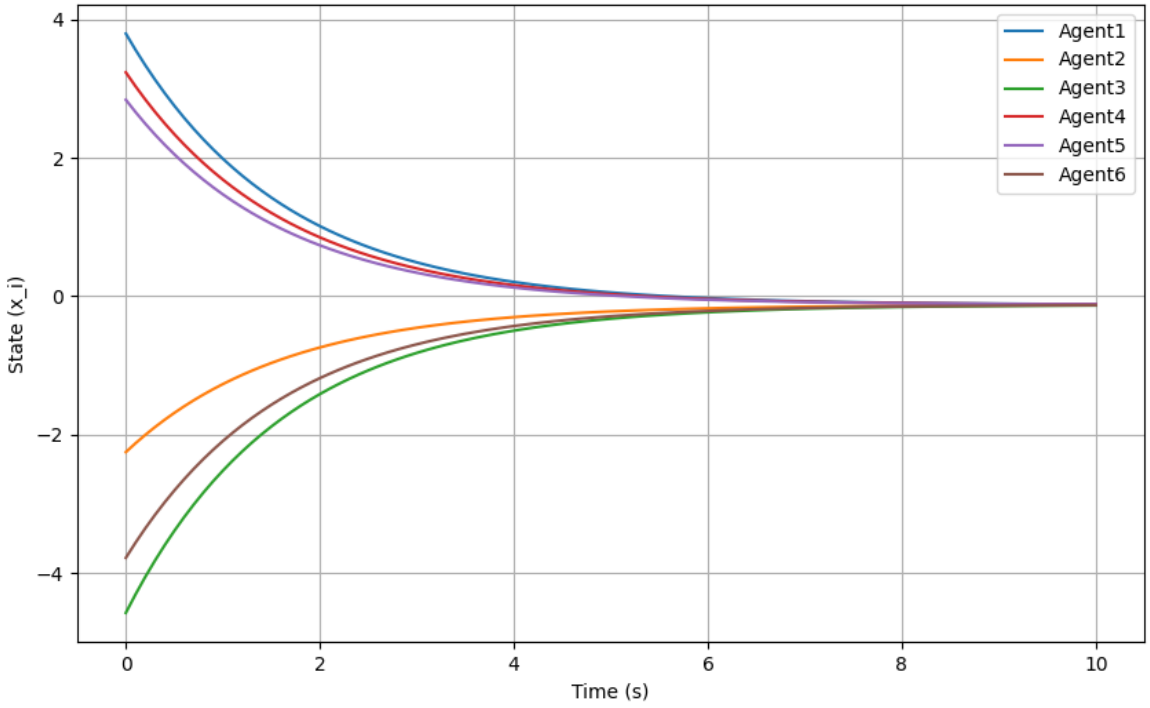


Figure 2.1: Leaderless Consensus of 6 Robots

2.2.2 Leader Consensus

In leader-follow, on the other hand, the leader agent sets the course of action for the other agents. To determine the leader's position, followers that is, other agents—communicate and exchange knowledge. Agents either jointly select the leader or it is predetermined [25].

A mobile leader or a team of agents serving as leaders are both possible in MAS. A mobile leader has the ability to relocate. As a result, agents might have to monitor the leader's location, which adds to the delays and overheads of processing and communication. When there are several leaders, they work together to lead the followers by exchanging ideas and making choices [10]. Consider a group of N robots and a leader robot denoted by $x_0(t)$ where the dynamics of each robot is given as a single integrator [23]:

$$\text{The Leader:} \quad \dot{x}_0(t) = u_0(t) \tag{2.6}$$

$$\text{The Followers:} \quad \dot{x}_i(t) = u_i(t) \tag{2.7}$$

where $i = 1, 2, \ldots, N$, $x_i(t) \in \mathbb{R}$ is the state of the i-th robot/agent and $u_i(t) \in \mathbb{R}$ is the control law to be determined.

The leader-following is said to be achieved if [26]

$$\lim_{t \to \infty} \|x_0(t) - x_i(t)\| = 0 \tag{2.8}$$

is satisfied for all $i \in \mathcal{V} \cup \{0\}$ and for all initial conditions.

In Figure 2.2, the leader's state is shown in blue, and the followers work together to follow it as it moves in a particular arrangement. The leader and every follower agent start moving under diverse and different initial conditions. Within three seconds, the leader's state is reached by the states of every agent.

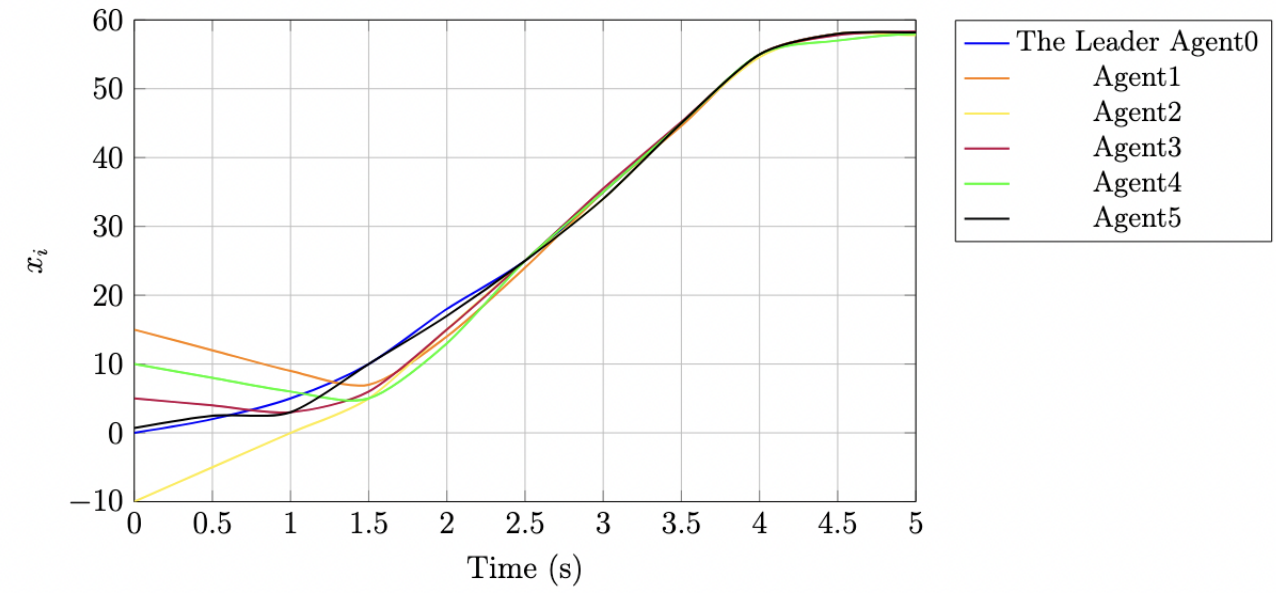


Figure 2.2: Leader-Following Consensus of 6 Robots

2.3 Formation Control

In coordinated control of a collection of unmanned autonomous vehicles or robots, formation control is a crucial challenge. A collection of self-driving cars must often adhere to a predetermined path while preserving a specific spatial arrangement. There are many benefits of moving in formation over traditional systems [27].

Applications for formation control are numerous, including security patrols and search and rescue missions in dangerous situations. For area coverage and surveillance, a group of autonomous vehicles must maintain a specific formation throughout military operations. Similarly, in tiny satellite clusters, formation helps to increase sensing capabilities [28].

The formation problem for the multiagent (2.4) is said to be solved, if [23]

$$\lim_{t\to\infty} \|[x_i(t) - x_j(t)] - [d_i - d_j]\| = 0 \tag{2.9}$$

is satisfied for all agents $i, j \in \mathcal{V}$ and for all initial conditions, where $d_i - d_j$ represent the displacement vector between the agents.

2.4 Mathematical Preliminaries

In this section, key concepts from graph theory relevant to our study is introduced, with a focus on connected and disconnected graphs.

2.4.1 Graph Theory

Vertices and edges, which represent the communication topology, have been widely described using graph theory. Assume that $G = (V, E, A)$ is a network with N nodes, where $V = \{1, 2, \ldots, N\}$ A set of vertices is denoted by V, a set of edges by an order pair $E \subseteq V \times V$ and the adjacency matrix $A = [a_{ij}]$ with non-negative entries.

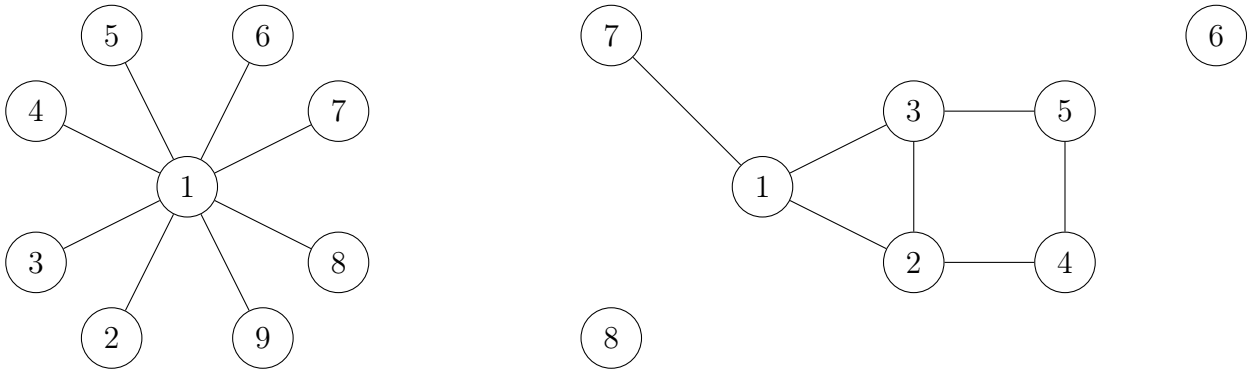
Let $N_i = \{j \in V \mid (j, i) \in E\}$ represent the set of neighbors of agent i. If and only if $a_{ij} = 1$ and zero otherwise, then $j \in V$, meaning agent j is one of i's neighbors [29]. So, a graph is a combination of a set of vertices and a set of edges which contain the vertices, each edge linking two of the vertices [23].

There are two types of Graphs: directed and undirected graphs. Undirected graphs show a mutual link between vertices since their edges lack direction, see Figure 2.3a. On the other hand, directed graphs are shown as ordered pairs of vertices with a direction as shown in Figure 2.3b.

(a) Undirected graph with 5 vertices (b) Directed graph with 4 vertices

Figure 2.3: Representation for Directed and Undirected Graphs

The graphs could be connected or disconnected. The connected graphs have a path connecting each pair of vertices. In a connected undirected graph, every node is reachable, forming the graph as a single, coherent unit [23]. As shown in Figure 2.4a. However, a graph is said to be disconnected if there are at least two vertices in a network without a path connecting them. These graphs are made up of two or more disjointed sub-graphs, which are called components [21], as shown in Figure 2.4b.



(a) Connected graph G (b) Disconnected graph G

Figure 2.4: Examples of Connected and Disconnected Graphs.

Laplacian Matrix

One well-known and important metric in graph theory is the Laplacian matrix. As it turns out, the properties of the Laplacian matrix vary depending on whether the graph is connected or disconnected [23]. This matrix provides insights into the graphs, regrading

their connectivity.

$$L = D - A \tag{2.10}$$

is the Laplacian matrix of graph G, where $D = diag(d_1, d_2, \ldots, d_N)$ *is the degree matrix and* $d_i = \sum_{j=1}^{N} a_{ij}$. Where D is the degree matrix, it is a matrix with diagonal structure where D_{ii} is the degree of the vertex i, $d_i = \sum_{j=1}^{N} a_{ij}$ [30]. In short, matrix of vertex degrees that is diagonal [30]. And A is the adjacency matrix [22]. The Laplacian matrix has the following properties [31]: It is a symmetric and positive semi-definite matrix. Which means all of its eigenvalues are real and positive. And the row sum of L is zero, meaning $L \cdot \mathbf{1} = 0$, where $\mathbf{1}$ is a vector of ones. As a result, it would make sense for the eigenvalues of a graph to be arranged in ascending order as [21]:

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n, \tag{2.11}$$

Where λ_i is the i-th eigenvalue of the Laplacian matrix $\mathcal{L}$ and $\lambda_1 = 0$. The eigenvalue $\lambda_1 = 0$ is always zero for all connected and disconnected graphs [31]. The second eigenvalue, λ_2, provides an indication on the connectivity of the graph [23].

The Laplacian matrix of a connected graph has various properties. The first value, which is the smallest eigenvalue of L is 0, and it has a multiplicity of 1 [22]. The eigenvector associated with the eigenvalue of 0 is the vector $\mathbf{1} = [1, 1, \ldots, 1]^T$ [22]. All other eigenvalues of L are strictly positive [30]. The second eigenvalue $\lambda_2 > 0$ indicates the graph is connected [23]. However The Laplacian matrix of a disconnected graph has $\lambda_2 = 0$ [30].

2.5 Positioning System

It is essential to consider that each agent's perspective of the MAS topology is restricted to its neighbors. Finding a specific agent, or localization can be difficult with this constrained vision.

Positioning systems that are used for monitoring and tracking involve the global position system (GPS) and the local position system (LPS). Because of GPS's limitations in indoor environments, the usage of LPS has become an absolute necessity for accurately estimating the position of objects. Choosing the ideal LPS system requires balancing accuracy, precision, coverage, and cost. This section provides an overview of LPS performance criteria, including current technologies, approaches, and methods [32].

2.5.1 Ultrasound Positioning Systems

Bats utilize ultrasonic signals to navigate at night, which inspired humans to develop a similar navigating method during the previous several centuries. The technology's performance is affected by obstructions and reflections between tags and receivers, which reduces the precision of the system. From a user's perspective, installing a lot of sensors on each room's ceiling takes a lot of time, which reduces the system's scalability. It is also necessary to locate the receivers precisely, which makes installation difficult and expensive [33].

2.5.2 Radio Frequency Identification Technology

RFID is a technique that uses electromagnetic transmission to an RF-compatible integrated circuit to store and retrieve data [34]. The advantage of an RFID locating system

is that individuals may be monitored by taking the tiny, light tags. Both the tracked individuals and the equipment can be uniquely identified by this technology. However, the working area of an RFID positioning system requires the installation and maintenance of a number of infrastructure components for proximity and absolute positioning approaches [33].

2.5.3 Bluetooth Technology

It is a wireless technique for estimating position. Its primary advantages are low power consumption and strong security. Also, because Tags transceivers are compact and do not require any infrastructure, they work well. However, the cost is higher and the coverage area is smaller [32].

2.5.4 Wi-Fi Technology

This approach is extensively used for local positioning since it can locate the position of an object or person well. Power consumption and signal attenuation from walls and doors are Wi-Fi's challenges, while dense deployment of wireless routers can increase accuracy [32].

2.5.5 Optical positioning systems

Robot self-localization in indoor environments is most often accomplished by optical indoor positioning [35]. Pre-installed markers have to be added to the system so that the camera detects a feature or a tag. Other than that, it should work well since this method is known for its flexibility and accuracy in indoor environments.

2.5.6 Ultra Wave Band

This technology offers various advantages over other positioning technologies used in the LPS, no multipath distortion and less interference. Thus, using UWB technology provides a higher accuracy [36].

This technology has the potential to interfere with current systems that employ the ultra-wide spectrum, as well as impact GPS and airplane navigation radio equipment.

2.6 Visual Markers for Localization

For robots to be able to figure out their position and orientation within an environment, a precise localization is necessary. Robots can navigate on their own and meaningfully engage with their environment with the aid of localization.

Visual markers, which are recognizable patterns positioned throughout the environment that a system can recognize and utilize to determine its location, are one of the best methods for achieving localization. These markers can be positioned on robots or throughout the environment.

Artificial visual characteristics known as fiducial are made for automatic detection and frequently have a special payload that helps them to be identified from one another [37]. Even though augmented reality programs were the ones that initially created and popularized these kinds of fiducial. They are made to be easily identifiable and differentiated from one another. Despite being connected to other 2D barcode systems, such as QR codes [38]. When using a QR code, a human usually has to align the camera with the tag and take a high-resolution picture of it, extracting hundreds of bytes, like a site address. A visual fiducial, on the other hand, has a tiny payload of information, perhaps 12 bits

but is made to be instantly recognized and located, even if it is not clearly visible due to noise, oriented strangely, or is at very low resolution. The alignment markers of a QR tag consist of roughly 268 pixels, whereas the majority of visual fiducial vary from roughly 49 to 100 pixels, which helps with long-range detection. Visual fiducial systems give the camera a relative location and orientation of a tag, in contrast to 2D barcode system where the barcode's position in the image is irrelevant. In addition, fiducial systems are made to identify several markers in a single picture [39].

There are different types of Visual Fiducial markers such as QR-code, ArUco Markers, and AprilTags. The QR code is a kind of two-dimensional or matrix bar code that is intended for smartphone reading and can contain data [40]. "Quick Response" or QR for short, means that the code's contents should be decoded rapidly. On a white background, the code is composed of black modules arranged in a square pattern [41]. Because of the poor orientation optimization, QR codes may be not visible when rotated or viewed from an angle that is not perpendicular to the screen. In robotics where pose estimation and orientation is essential, this renders QR codes less dependable.

ArUco markers are square markers in black and white that are used to reliably estimate camera posture. They are frequently used to track the direction and position of a camera or robot in augmented reality [42].

AprilTags are Visual fiducial markers that are made for effective pose estimation and detection. They consist of a square binary matrix and have an encoded pattern on it [39]. Relative to ArUco markers, AprilTags possess higher robustness, improved pose estimate performance, and faster detection along with the feature of error correction, making AprilTags perfect for those applications where accurate positioning is critical, notably in real-world scenarios and large scale systems [39].

Chapter 3

METHODOLOGY

Contents

This chapter contains a detailed description of the hardware components of the project and the software used. The design and architecture of the entire system, the methods and techniques utilized for control, communication, and other integrated systems, including localization. In addition, an illustration of how the system is built to accomplish its objectives is provided, from the hardware components to the software that connects everything.

3.1 Hardware Design and Components of Robots

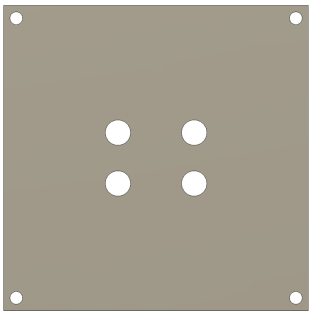
The design of these robots demanded a challenging process of selecting hardware elements that would increase the robots' efficiency, reconfiguration, and ease of coordination. A closer look at each robot's main components is presented.

3.1.1 Robot Platform

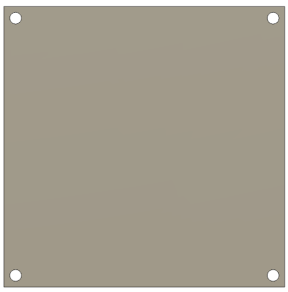
The body/base of each robot is made of a rigid yet lightweight material that also serves to maintain all components of the robots. The aluminum parts were used to build the robot so that it could move easily in any direction on the ground.

The robots' platform is shown in Figure 3.1. It has two plates to which the wheels and motors are fastened in between them and one plate to which all sensors and other components are attached.

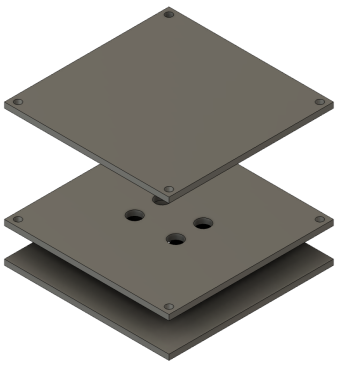
(a) Lower Plate



(b) Second Lower Plate



(c) Top Plate



(d) Side View of the Whole Platform

Figure 3.1: The Platform of the Robots

3.1.2 Micro-controller

The microcontroller is what operates each robot; in this project, an Arduino Uno in Figure 3.2 will be utilized for each robot. This is the robot's control center, where it manages all movement, coordinates communication, processes data, and operates all other components connected with it. More on different Arduino boards [43].

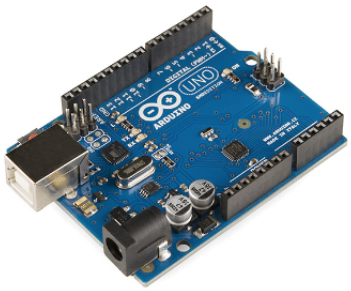


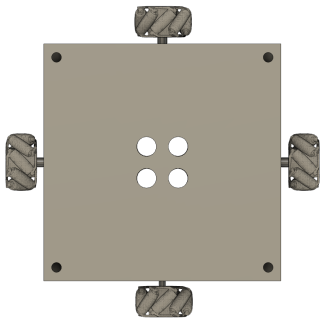
Figure 3.2: Arduino Uno

3.1.3 Mecanum wheels

The most distinctive characteristic of these robots is their mecanum wheels, also known as omni wheels, which allow for omnidirectional movement [44]. There are three degrees of freedom of motion for every Mecanum wheel also they have rollers that are angled differently than the standard wheels, which would enable the robots to move in any direction [45]. Figure 3.5 shows the wheel.

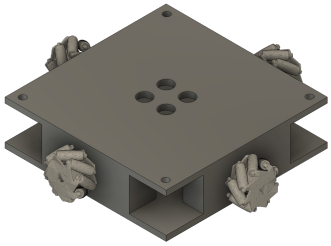
Figure 3.3: Mecanum Wheel

Figure 3.4 shows the design of the robot where 4 mecanum wheels are attached to the base and also shows the arrangement of the wheels.



(a) Top View of the Base

(b) Front View of the Base



(c) Side View of Robot Body

Figure 3.4: The Design of Robot Base Attached to Mecanum Wheels with Different Views

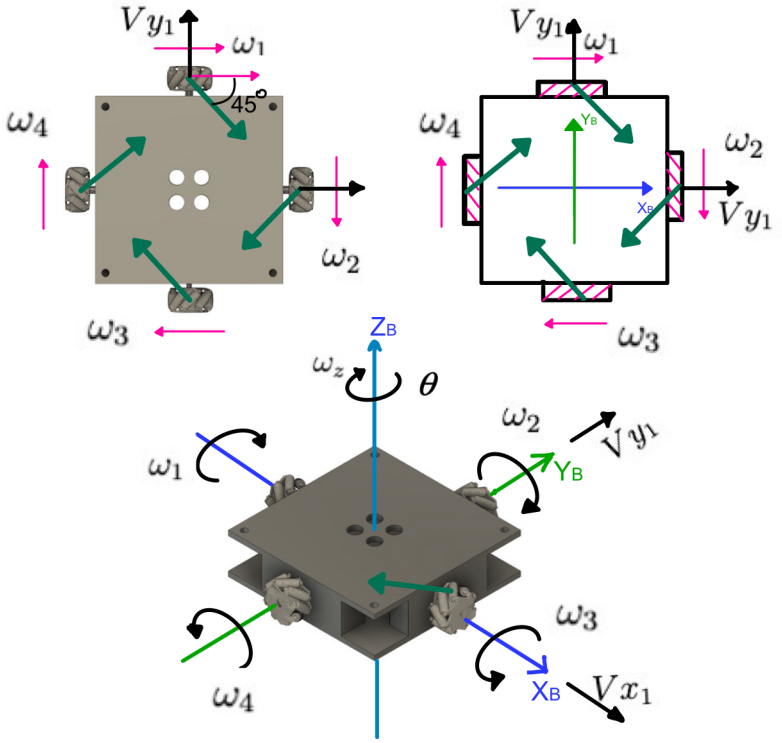


Figure 3.5: Coordinate system assignments for the four-Mecanum-wheel robot with the symmetrical proposed structure

3.1.3.1 Forward and inverse mechanics

According to the kinematics analysis of a single mecanum wheel, the relationship between the angular speed of the motors $\omega_1(t)$, $\omega_2(t)$, $\omega_2(t)$, $\omega_2(t)$ and the body frame velocity $Vx(t)$, Vy, ω_z can be determined as follows:

$$Vx = \omega_1 + \omega_2 - \omega_3 - \omega_4 \tag{3.1}$$

$$Vy = \omega_1 - \omega_2 - \omega_3 + \omega_4 \tag{3.2}$$

$$\omega_z = \omega_1 + \omega_2 + \omega_3 + \omega_4 \tag{3.3}$$

where ω_1, ω_2, ω_3, and ω_4 are the angular speeds of the motors. Vx, Vy, and ω_z represent the linear and angular velocities of the robot in the body frame perspective. These equations describe the forward kinematics. The forward kinematics is useful when determining the robot's translational and rotational velocities in the body frame, given the individual motor speeds. This analysis is essential to estimate the instantaneous motion of the robot during operation.

Inverse kinematics, on the other hand, is required to determine the individual wheel speeds $\omega_1, \omega_2, \omega_3, \omega_4$ needed to produce a desired motion in the robot's body frame, characterized by V_x, V_y, and ω_z. These equations are essential for motion control tasks, allowing the mapping of commanded translational and rotational velocities to the specific wheel speeds that the drive system must generate. The corresponding inverse kinematics relationships are given as follows:

$$\omega_1 = \frac{1}{4}\Big[Vx + Vy + Vz\Big] \tag{3.4}$$

$$\omega_2 = \frac{1}{4}\Big[Vx - Vy + Vz\Big] \tag{3.5}$$

$$\omega_3 = \frac{1}{4}\Big[-Vx - Vy + Vz\Big] \tag{3.6}$$

$$\omega_4 = \frac{1}{4}\Big[-Vx + Vy + Vz\Big] \tag{3.7}$$

which can be written in matrix form as:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ -1 & -1 & 1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix} \tag{3.8}$$

This formulation provides the explicit mapping from the desired linear and angular velocities in the body frame to the required wheel angular velocities.

The previous analysis is dedicated to the robots equiped with Right-Mecanum wheels. For the Left-Mecanum wheels, we can repeat the same process to get the following inverse kinematics:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix}$$

3.1.3.2 Velocity Transformation

When the controller computes the control signals, it generates the desired linear and angular velocities of the robot in the inertial (global) frame, denoted by $(V_x^I, V_y^I, \omega_z^I)$. However, since the wheel velocities and robot kinematics are defined in the body frame, it is necessary to transform these inertial velocities to the body frame before computing the required motor speeds.

The transformation between the inertial frame and the body frame can be expressed using the rotation matrix $R(\theta)$:

$$\begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x^I \\ V_y^I \\ \omega_z \end{bmatrix} \tag{3.9}$$

where θ represents the orientation of the robot with respect to the inertial frame. This transformation is critical to ensure that the computed wheel speeds are consistent with the robot's kinematic constraints in its local frame. Thus, transforming the inertial frame velocities to the body frame guarantees that the control signals produce the intended motion relative to the robot's current heading.

3.1.4 Motors and Motor Driver

To move each wheel, every robot has been equipped with four DC gear motors shown in Figure 3.6a. Where a motor driver L293D in Figure 3.6b controls each motor. The rotational speed and direction of each wheel are set by these drivers.

The motor driver shield shown in Figure 3.6b uses two L293D ICs, each of which can

control two motors. This allows the driver to handle a total of four motors, which is just what the robots require. So, each robot will be equipped with four DC gear motors and a motor driver, since it can control all four motors.

(a) DC Gear Motor

(b) L293D Motor Driver Shield

Figure 3.6: Motor and Motor Driver Shield Used For The Robots

Figure 3.7 shows how all of the motors and the driver are connected with the Arduino Uno. It's important to point out, that the circuit schematic shows two L293D ICs, which are equivalent to the L293D motor driver shield used in the robots.

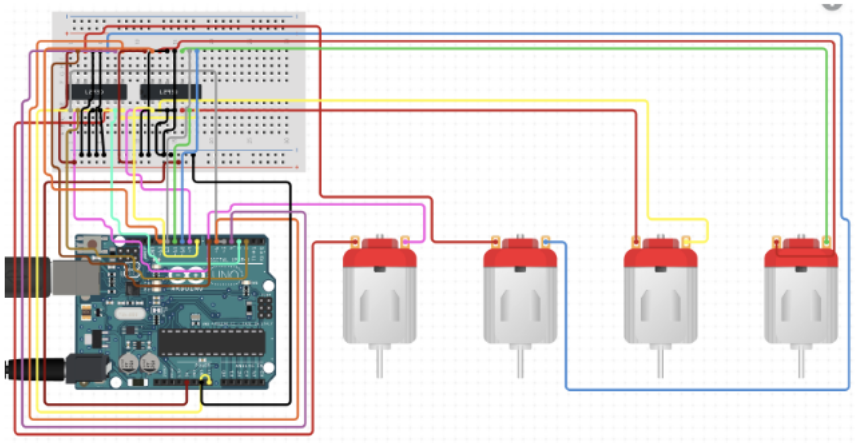


Figure 3.7: Connections of All Four Motors and a Motor Driver Sheild with Arduino Uno.

3.1.5 Communication

The primary reason wired communication is not employed in this project is that it does not meet the system's requirements. Cables may slow down the robots and influence their range and direction of motion, which is a disadvantage when the robot needs to travel and carry out certain tasks smoothly without a cable. In addition, it can be difficult to manage cables and connectors since the system becomes more complicated when working with multiple robots. Wireless communication, therefore, works best in such scenario. It provides the flexibility that these robots need to work smoothly.

In robotics systems, communication is essential for unlimited, targeted interaction and cooperation between robots and their surroundings. The robots' ability to communicate with one another can increase their effectiveness and capacities [46]. The two types of connection technologies are wireless and wired connections. Compared to wired communication, which includes cabled systems like Ethernet or serial connections, wireless communication technologies, such as Wi-Fi, Bluetooth, ZigBee, RF, and others, are more adaptable and scalable. Every communication method offers benefits depending on the situation.

In this project, the nRF24L01 wireless module shown in Figure 3.8, will be used and integrated into each robot so that they can exchange data and share information about their movements and positions. This will enable each robot to make its own choices, which will result in a cooperative multi-robot system. This module was chosen because it is energy efficient, provides communication over long range and transfers data quickly, making it a great fit for this project. Unlike Wi-Fi or Bluetooth, it offers a more stable and effective way for the robots to communicate in real time, which is especially important

when several robots need to cooperate and work together.

Figure 3.8: nRF24L01 Wireless Module For Communication

Wiring connections between nRF24L01 and Arduino Uno are illustrated in Figure 3.9 and table 3.1.

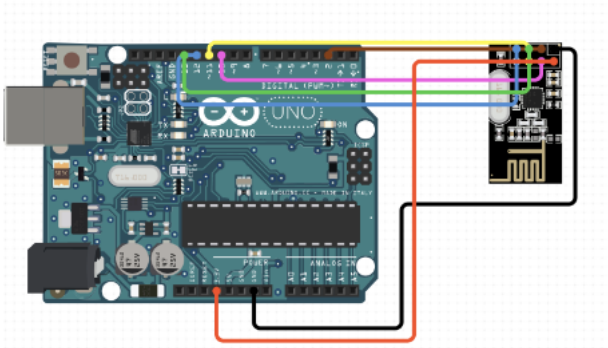


Figure 3.9: Circuit Diagram for nRF and Arduino Uno

nRF24L01 Pin	**Arduino Uno Pin**
GND	GND
VCC	3.3V
CE	Pin A5
CSN	Pin A4
SCK	Pin 13
MOSI	Pin 11
MISO	Pin 12

Table 3.1: Wiring Connections Between nRF24L01 and Arduino Uno

3.1.6 Power Supply

A Type-C lithium battery module with a built-in DC-DC step-up converter powers each robot using two rechargeable lithium batteries, each of which provides 4.2 volts. It boosts the voltage to a stable 12V output and is designed to keep the robots running smoothly without interruption until their tasks are completed. The battery is connected to the Arduino Uno by wiring it to the ground and input voltage pins, the battery module is shown in fig 3.10.

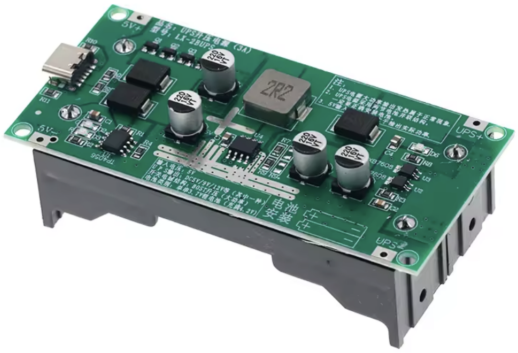


Figure 3.10: Type-C Lithium Battery Module - Step-Up Booster

3.1.7 Complete Robot Wiring Connections

This section explains how all the components that were shown in the previous sections came together on an Arduino Uno. Figure 3.11 illustrates the entire wiring setup, which includes the following key components:

- Arduino Uno: microcontroller that manage all operations of the robots.
- L293D motor shield driver: controls all four motors.
- Motors: each mecanum wheel is attached to a geared motor.

- nRF24l01 Module: enables wireless communication
- Batteries: provides energy to the motors.

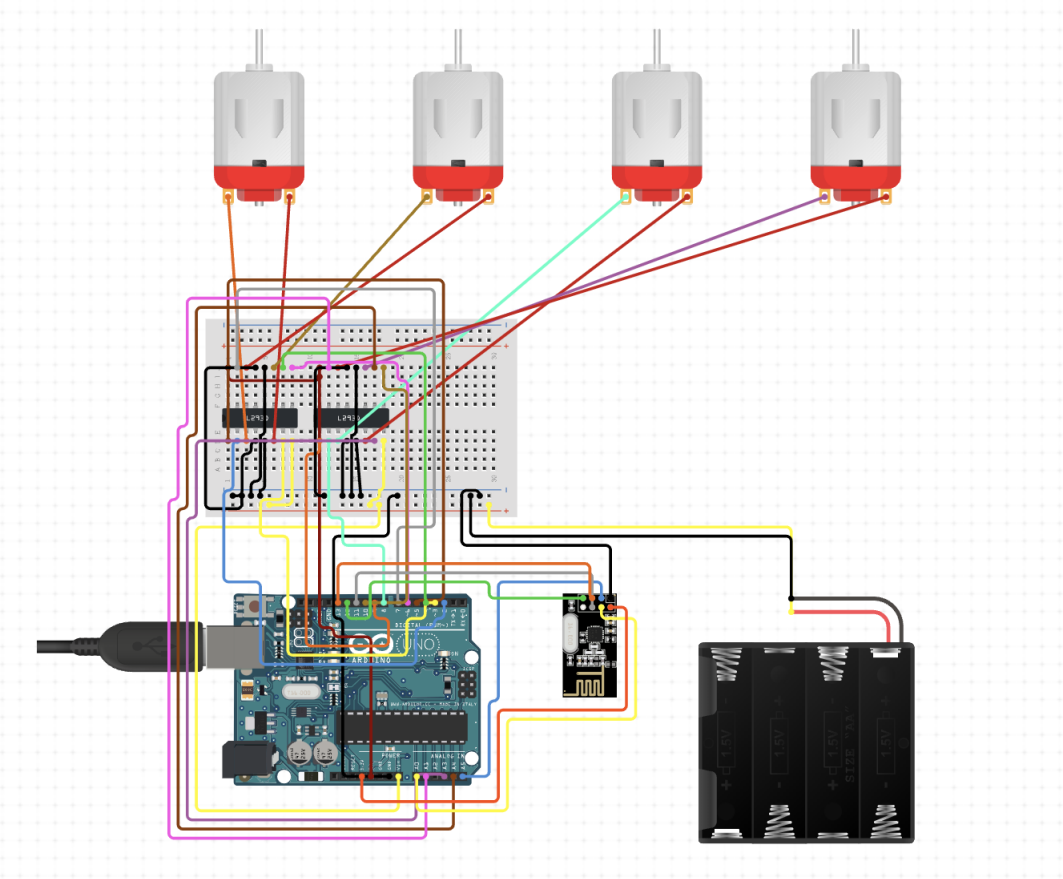


Figure 3.11: Complete Wiring of All Hardware Components for Each Robot

3.2 Localization

For this entire system to function properly, localization is essential. It is how the robots can determine each one's precise location and orientation, enabling precise movement and cooperation.

This localization system uses AprilTags and an overhead camera, which is a simple yet powerful setup. The camera is located two meters above the ground. The AprilTags, which are distinctive markers attached to every robot, simplify the recognition and tracking of

the robots by assigning a number to each one, which in the end is nothing more than a walking AprilTag for the camera.

3.2.1 The Camera

A high-resolution camera has been placed above the workspace to capture everything that is happening below. It gives a full view of where the robots are and how they're moving on the entire workspace. The camera used in the localization system is shown in Figure 3.12.The resolution of it is up to 2K at 30 frames per second (fps). The 2K resolution of the camera at 30 frames per second is a key factor in making the localization system work effectively. Hence, it guarantees that AprilTags can be recognized correctly even if they are small, located at a far distance, or a bit hidden. This level of detail allows us to track multiple robots clearly across a wide area.

Figure 3.12: Camera Used for Localization

3.2.2 AprilTags

For the purpose of this project, visual markers are crucial instruments for tracking and placement of the robot within the formation control system. Each robot is equipped with its own AprilTag, which works as a unique barcode that the camera can quickly and easily recognize. These tags are incredibly useful because they're fast to detect

and provide accurate information about the robot's position and orientation. Using the AprilTag library, the system processes the camera feed in real-time, pinpointing where each robot is and which way it is facing with impressive precision.

AprilTags are a square, black and white tag with an encoded binary payload [37]. They come in different families; each is made for specific needs based on its size and complexity. Some examples include 'tag16h5', 'tag25h9', and 'tag36h11'. For this project, the tag16h5 family is used (See Figure 3.13). These tags use 16-bit encoding, which makes them small and efficient while still being reliable enough to detect accurately, even if there is a bit of interference or noise. This balance makes them perfect for our system, allowing us to track the robots in real time without putting too much strain on the processing.

Figure 3.13: AprilTag of Family 16h5

3.3 Control System

The use of PID control is central to achieving desired system behavior. PID (Proportional-Integral-Derivative) controllers were implemented to regulate each robot's motion and coordination within the multi-robots framework. The PID control approach was selected for its simplicity, robustness, and effectiveness in handling dynamic system responses and disturbances.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

The parameters K_p, K_i and K_d represent the proportional, integral and derivative gains of the PID controller, respectively.

- The proportional gain K_p determines the magnitude of the response to the current consensus error e, providing an immediate corrective action proportional to the error value.
- The integral gain K_i addresses accumulated past errors by integrating the consensus error over time, which helps eliminate steady-state offsets and improves long term accuracy.
- The derivative gain K_d predicts future trends of the consensus error considering its rate of change, enhancing system stability and reducing overshoot.

Careful tuning of these gains is essential to achieve an effective balance between responsiveness, accuracy, and stability in the multi-robots formation control.

$e(t)$ represents the consensus error, which quantifies the deviation between the state of an individual agent and the desired consensus/formation state within the multi-robots system which can be written as follows:

$$e(t) = (\mathcal{L} \otimes I)\Big(x(t) - r(t)\Big)$$

where $x(t) = [x_1(t),\ y_1(t),\ \theta_1(t),\ x_2(t),\ y_2(t), \theta_2(t),\ \ldots,\ x_6(t),\ y_6(t),\ \theta_6(t)]$

$r = [rx_1,\ ry_1,\ rz_1,\ rx_2,\ ry_2,\ rz_2,\ \ldots,\ rx_6,\ ry_6,\ rz_6]$

Let the Laplacian matrix $\mathcal{L}$ is partitioned as follows

$$\mathcal{L} = \begin{bmatrix} \mathcal{L}_1 \\ \vdots \\ \mathcal{L}_6 \end{bmatrix}$$

The following defines the error in the coordinates x, y and θ for the first robot:

$$error_{x_1} = \Big(\mathcal{L}_1 \otimes [1,\ 0,\ 0]\Big)\Big(x(t) - r\Big)$$

$$error_{y_1} = \Big(\mathcal{L}_1 \otimes [0,\ 1,\ 0]\Big)\Big(x(t) - r\Big)$$

$$error_{z_1} = \Big(\mathcal{L}_1 \otimes [0,\ 0,\ 1]\Big)\Big(x(t) - r\Big)$$

To implement the PID controller, $\Big(\mathcal{L}_1 \otimes [1,\ 0,\ 0]\Big)x(t)$ is used as the actual value of the state and $\Big(\mathcal{L}_1 \otimes [1,\ 0,\ 0]\Big)r$ is used as the set point or the desired value of the state.

The following sample from the code explains how the PID performs these operations component.

```
I1=np.array([  [1, 0, 0] ])
```

```
pid_x1.setpoint = np.kron(L[0,:],I1)*r
```

```
Lx1 = np.kron(L[0, :], I1).flatten() @ X.flatten()
```

```
Vx1 = pid_x1.update(Lx1, dt)
```

Where Vx_1 is the output of the PID controller to regulate the $x-$axis speed of the first robot in the inertial frame. This operation must be performed for x, y and z for all robots.

It should be noted that Vx_1 is the speed of the robot in the inertial frame. However, the robots are controlled in the body frame. To solve this problem, rotation matrices are used to convert the speed in the inertial frame into the body frame using Equation (3.9). Following this, the robot's overall velocity must be transformed into the corresponding motor velocities using the inverse kinematics matrices given in (3.8).

3.4 Software Framework

The software is what happens behind the scenes to make robots effective and intelligent. The achievement of the ideal mix of scalability, performance, and simplicity was essential for the development of the software framework. This is how it all came together:

3.4.1 Programming Tools

The main programming software used for the robots are Python and Arduino IDE. The Arduino IDE is ideal for controlling the micro-controllers and hardware, while Python handles more complicated areas, such as localization. These powerful, user-friendly tools come with lots of libraries and support systems that enhance rapid development.

The main programming tools used in this project are Python and the Arduino IDE. Python is responsible for managing the overall system logic, including localization, formation control, and the coordination needed for the six robots to cooperate. On the other hand, the Arduino IDE is used to program each robot individually, allowing them to receive data from the communication module and respond accordingly. Additionally, a separate Arduino script handles data transmission. For more implementation details, including the system-level Python code and the Arduino scripts for both receiving and transmitting data, refer to appendix .2 .3 .4.

3.4.2 Simulation Tools

The action performed by the robots is analyzed and simulated within the context of a simulation environment before being deployed into the real world. For modeling the system, tools like Simulink and MATLAB are used. A detailed explanation of the MATLAB-based simulation for different behaviors of a multi-robot system can be found in chapter 4.

3.4.3 Computer Vision and AI

Computer vision helps robots recognize and understand what is happening in the world, or in other words, to "see". In this case, libraries such as OpenCV and AprilTag make most of the effort. For example, to detect robots in the localization system, there are AprilTags in the environment, and AI models trained in item detection to help the camera recognize an object. These libraries are solid and relieve us from the need to begin from scratch, enabling us to focus on how the technology is used.

3.4.4 3D Sketching

For the purpose of this project, multiple design iterations were required. Fusion 360 was used to create 3D sketches and design the robots' platform. The final robot design is illustrated in Section 3.1.1.

Chapter 4

SIMULATION AND ANALYSIS

Contents

A simulation of the multi-robot system in Matlab is offered in this chapter to guarantee that everything will be coordinated and smooth when it comes to the actual workspace where they do their actual tasks. In this manner, a simulation test of the response is performed without endangering the hardware. Simulink and MATLAB tools are being utilized to simulate the system. Afterward, once the simulation is good enough, will be moving on to physical tests and prototyping. Different behaviors were tested before the actual graph, exploring various approaches and interactions for the multi-robot system.

4.1 Multiple Simulation of Multi-Robot System Connections

This chapter examines the behavior of a multi-robot system in four distinct connection configurations: fully connected, partially connected graph, and disconnected graph. These examples show how the system's performance and results are affected by the level of connectivity. A simulation for every setup is implemented using Simulink. An illustration of each model is presented to demonstrate the connections between the robots and the outputs that were obtained.

In each simulation of a graph model, each node represents a robot, and the links represent the connections among the robots. As for each system output for a graph, it shows if the system reached a consensus or not.

4.1.1 Complete Connected Graph

Every robot in a completely connected graph is directly connected with every other robot, which makes all robots capable of communicating with each other. An example of a fully connected graph is shown in 4.1 which consists of six robots. This high degree of connectivity makes coordination easy and significantly reduces the possibility that communication breakdowns would affect the functionality of the system. As demonstrated by this approach, robots have the ability to perform tasks effectively and with the best coordination. Furthermore, it seems quite resilient to single robot failures, making it a robust system in which the failure of one robot does not affect the network as a whole.

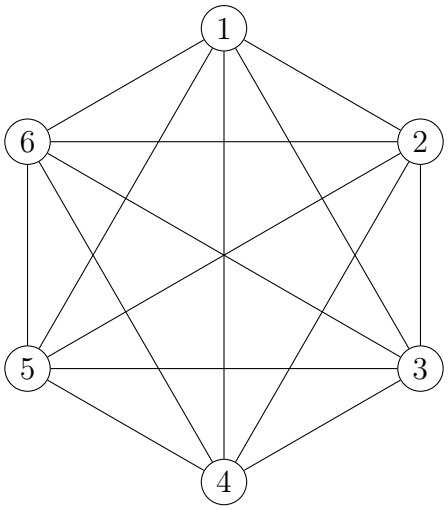
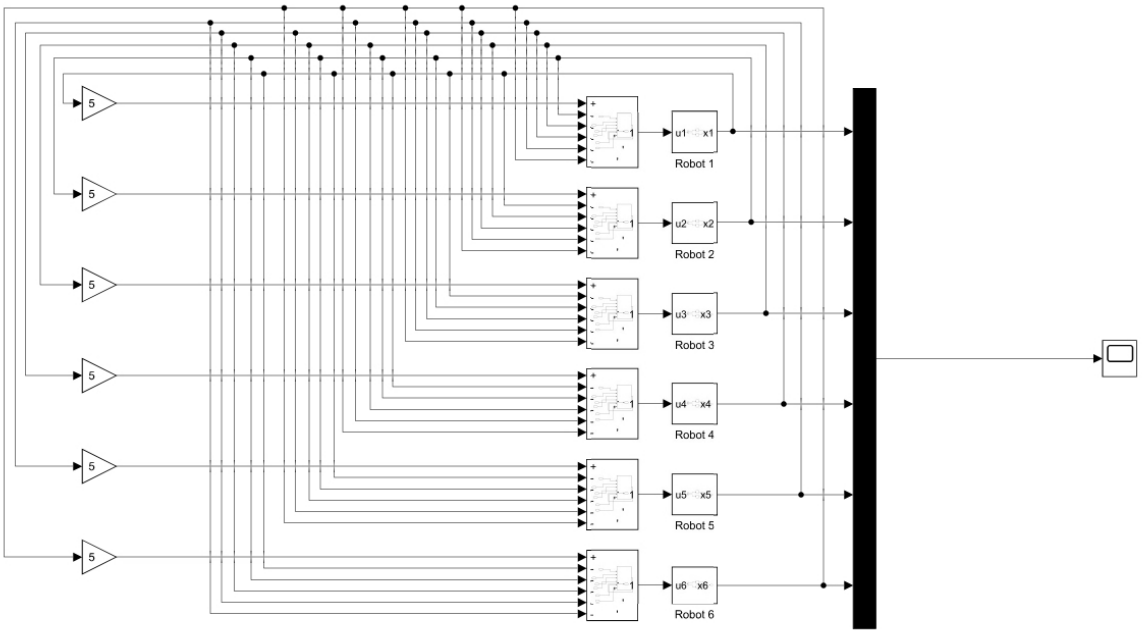
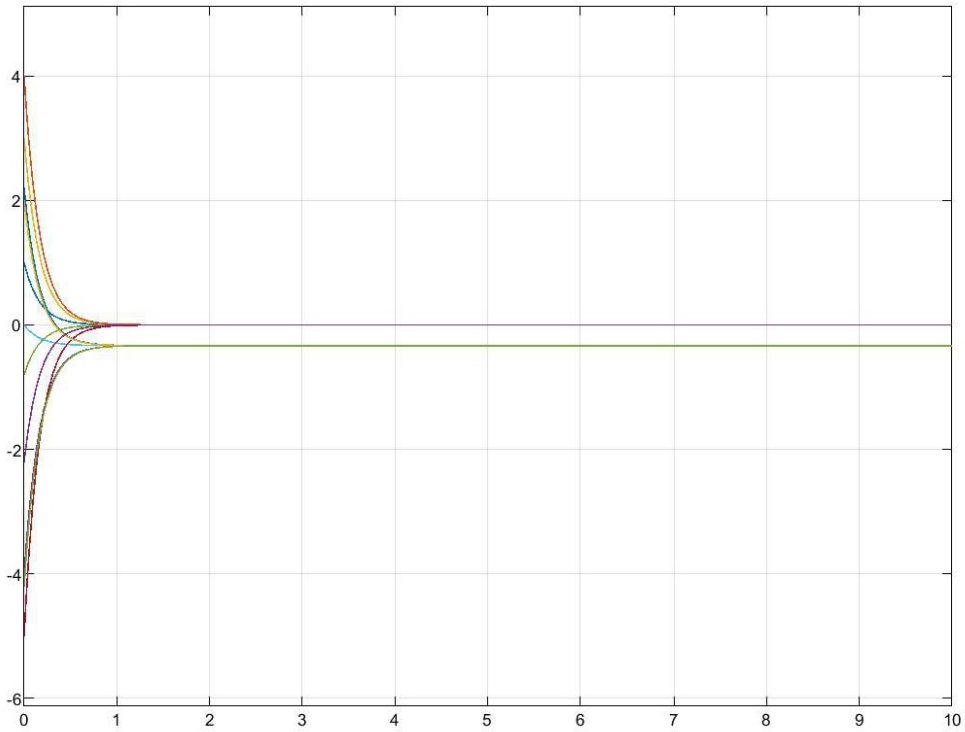


Figure 4.1: Fully Connected Graph of 6 Robots

Figure 4.2 shows the simulation of the fully connected graph shown in 4.1, all robots reach consensus within the first second. All x-values for each robot reaches a certain point and the same for y-values.



(a) Model Representation of a Complete Connected Graph of 6 Robots



(b) Output Result of a Complete Connected Graph of 6 Robots

Figure 4.2: Connections and Output of a Fully Connected Multi-robot System.

4.1.2 Partially Connected Graph

A network with fewer connections between nodes is created when a part of the robots are able to interact directly, as shown by a partially connected graph in figure 4.3. In contrast to the previously displayed fully connected graph, not every robot is a neighbor in this scenario. This configuration is more realistic since it takes into account actual constraints like a limited communication range or obstructions that prevent connections. This system might occasionally be vulnerable to failure. However, these systems often function well, but there might be occasional delays in solving the consensus/ formation problem.

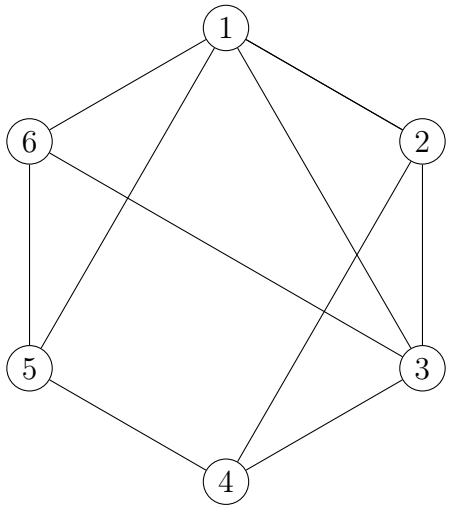
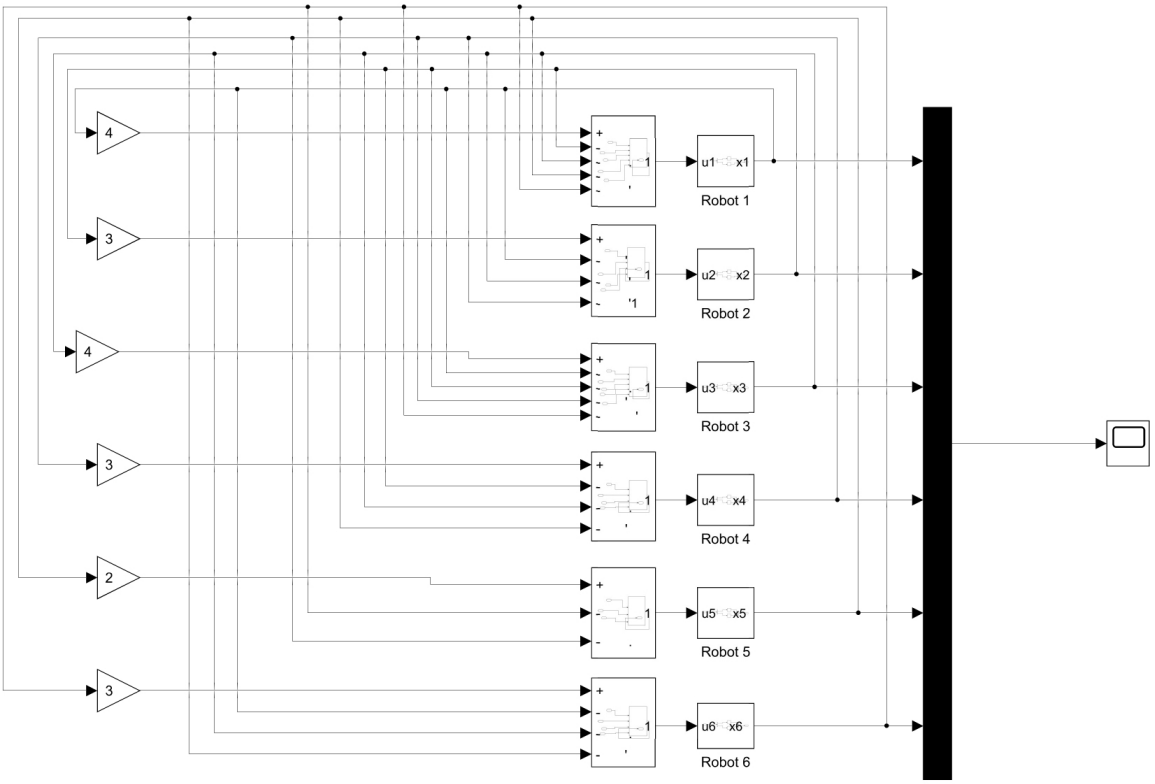
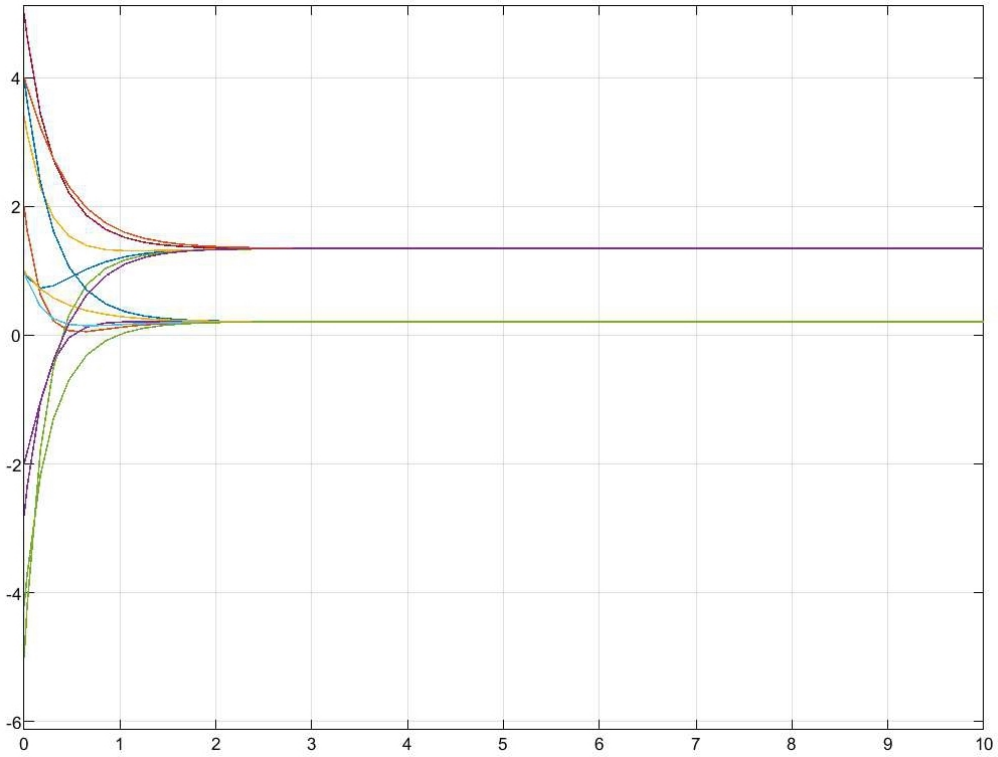


Figure 4.3: Connected Graph of 6 Robots

The simulation in Figure 4.4 shows that the robots can effectively coordinate and reach consensus in two seconds, even with fewer connections than the completely connected graph, which was the previous example. The result shows that the x-axis values align at one point, while all the y-axis values for every robot come together at another. This demonstrates the effectiveness of the system's communication and control techniques by highlighting its ability to cooperate and reach consensus despite the difficulties caused by partial connectivity.



(a) Model Representation of Connected Graph of 6 Robots



(b) Output Result of a Connected Graph of 6 Robots

Figure 4.4: Connections and Output of a Consensus Connected Multi-robot System.

4.1.3 Disconnected Graph

Robot functionality in a disconnected network depends on the graph and which robots are disconnected. In this scenario, which is shown in 4.5, robots 1 and 2 will operate independently without contacting any other robot because they are connected to one another but completely separated from the others. In this most constrained situation, there is no coordination or contact between the robots; instead, each one is entirely dependent on its local sensors or pre-programmed instructions. It's important to note that poor communication results in an inefficient system where a possibility of robot collision arises. This concludes that this system is the worst of the scenarios that have been previously demonstrated because it does not function well nor reach consensus.

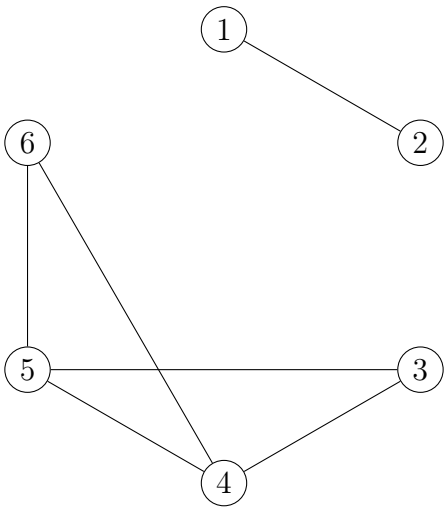
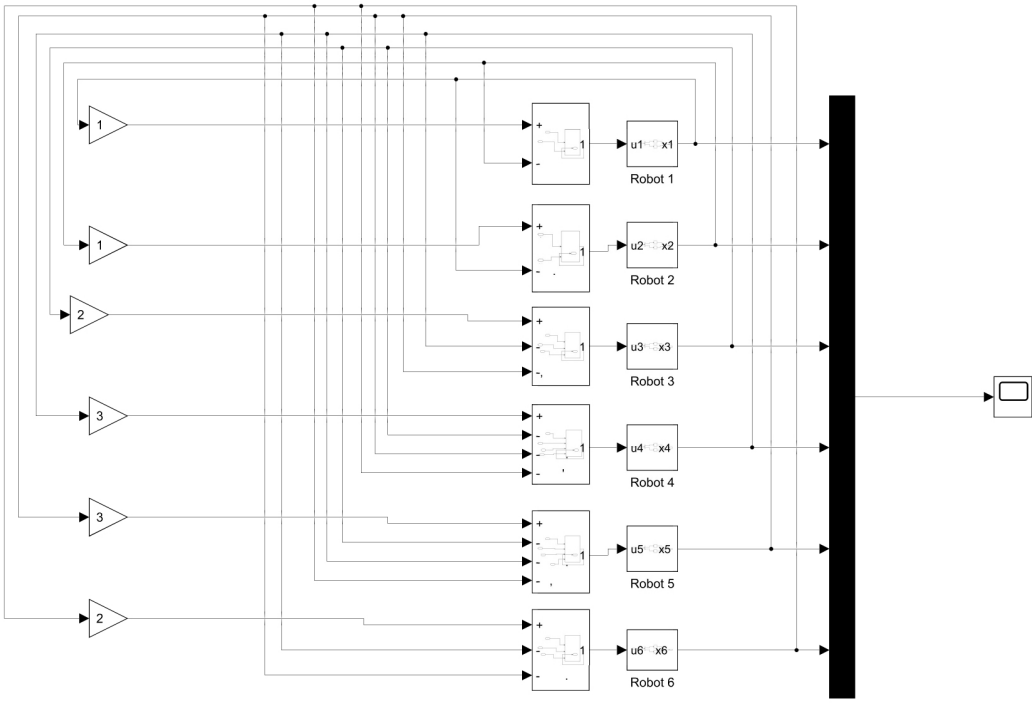


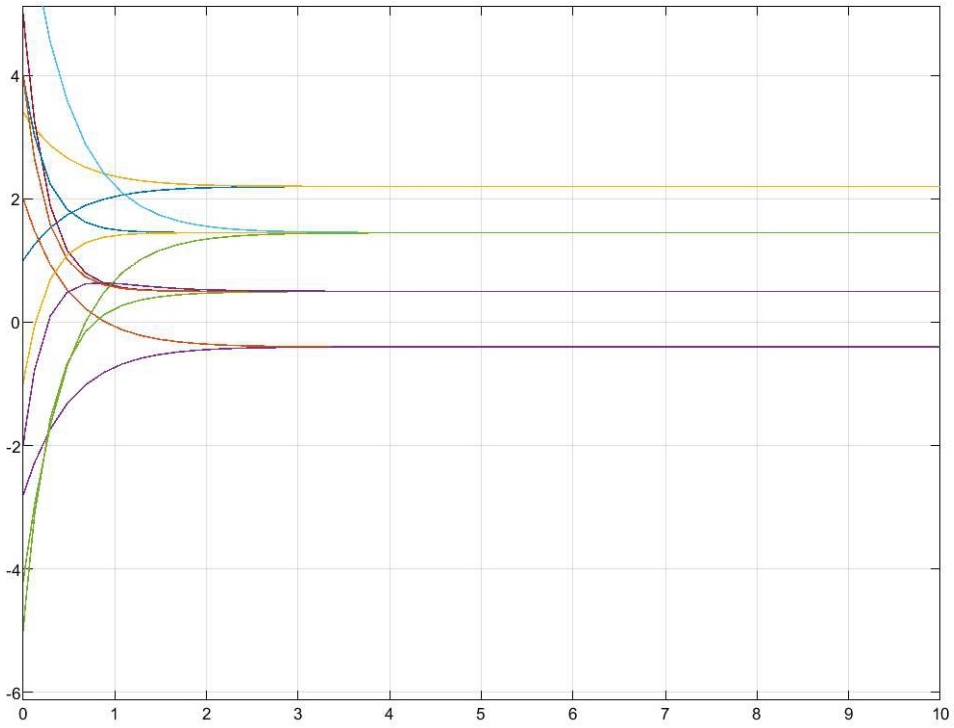
Figure 4.5: Disconnected Graph of 6 Robots

Since robots 1 and 2 are disconnected from the other robots that are coming to an agreement, the simulation in 4.6 shows that the robots are unable to reach consensus. Thus, with the appearance of disconnectivity the outcome demonstrates that while all of the x-axis values for the connected robots come together at one point, the y-axis values for the same connected robots align at another. However, the x and y values for the discon-

nected robots are unable to do the same. Given that robots were unable to collaborate and reach consensus, this indicates that the system's effectiveness is extremely poor.



(a) Model Representation of Disconnected Graph of 6 Robots



(b) Output Result of a Disconnected Graph of 6 Robots

Figure 4.6: Connections and Output of a Disconnected Graph.

In conclusion, the importance of communication in multi-robot systems is highlighted by the simulation of these three connectivity scenarios. Complete graphs facilitate smooth coordination and reached consensus, whereas partially connected graphs balance resource constraints with performance. In contrast, disconnected graphs show how hard it is for robots to perform without coordination. These findings emphasize that in order to guarantee the success of multi-robot systems, robust and efficient communication techniques must be developed.

4.2 Simulation Results of the Proposed Graph

This section focuses on the graph that was developed as part of this project. The purpose of this simulation is to validate the behavior of the graph. A visual representation and results of the robots coming to an agreement and their trajectory are provided. This simulation successfully demonstrated the structure and functionality of the communication edges between the six robots, providing insight into their performance. This step is important in the overall analysis to ensure an accurate and reliable foundation for implementation.

Figure 4.7 shows the implemented graph in this project and the edges represent the communication links between the robots.

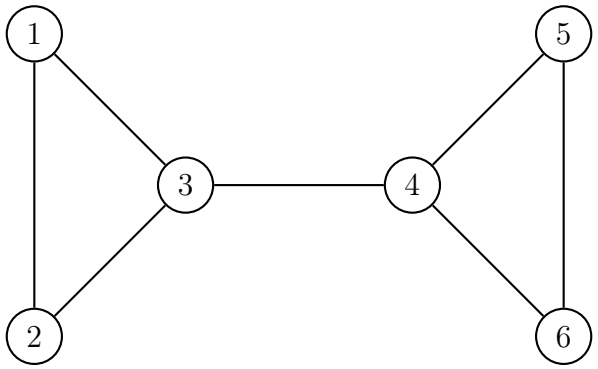
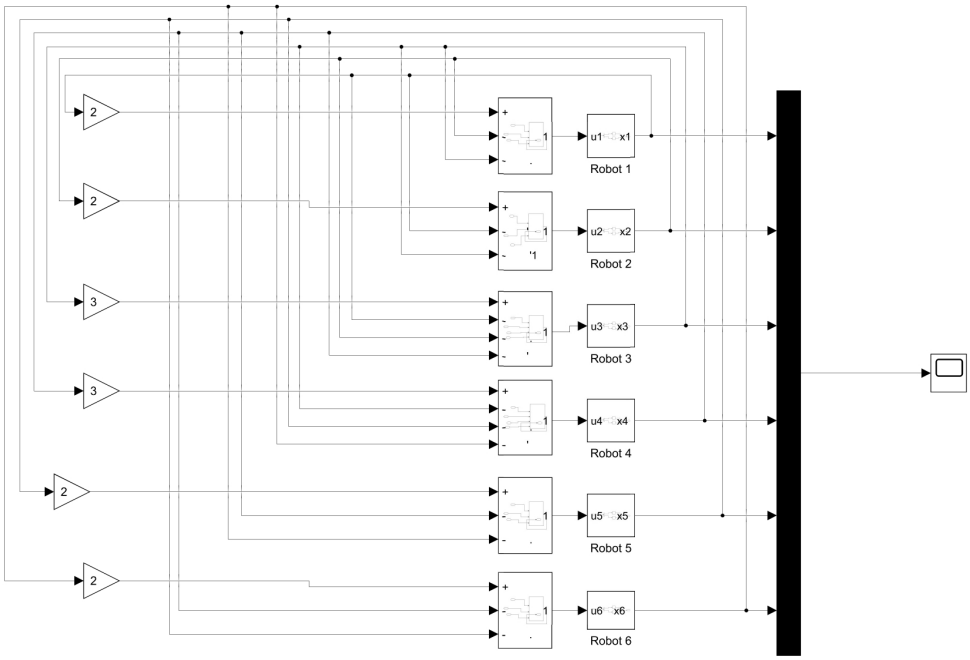
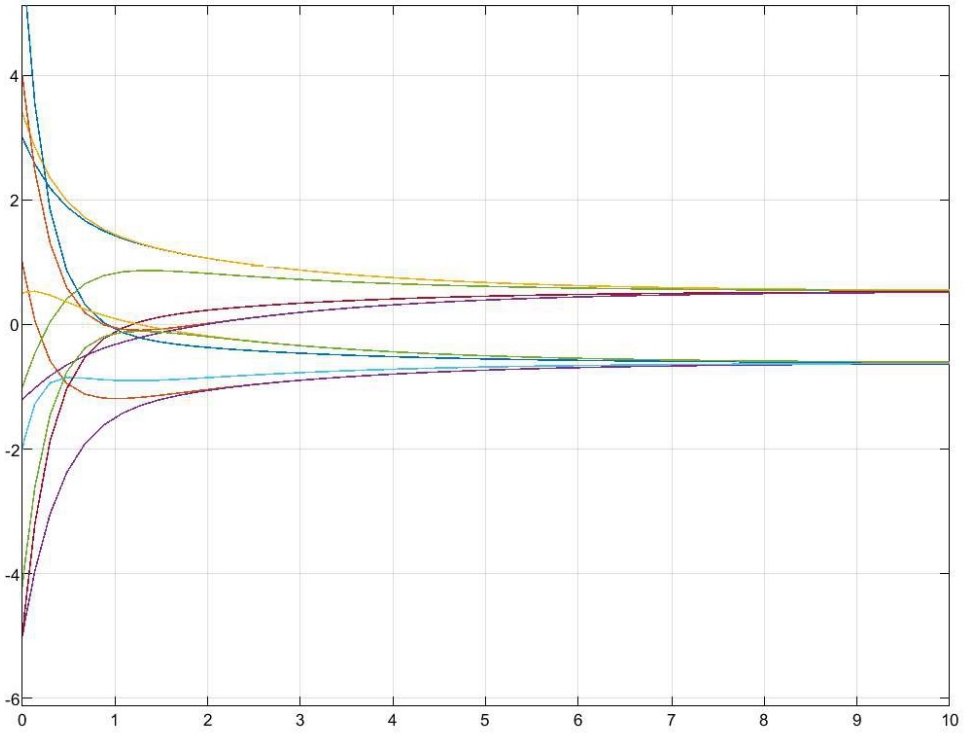


Figure 4.7: The implemented Connected Graph of 6 Robots for this Project

Simulation of the graph in Figure 4.7 is provided:



(a) Model Representation of the connected graph.



(b) Output Result of 6 robots reaching consensus Within 9 Seconds.

Figure 4.8: Simulation of the Implemented Graph for this Project.

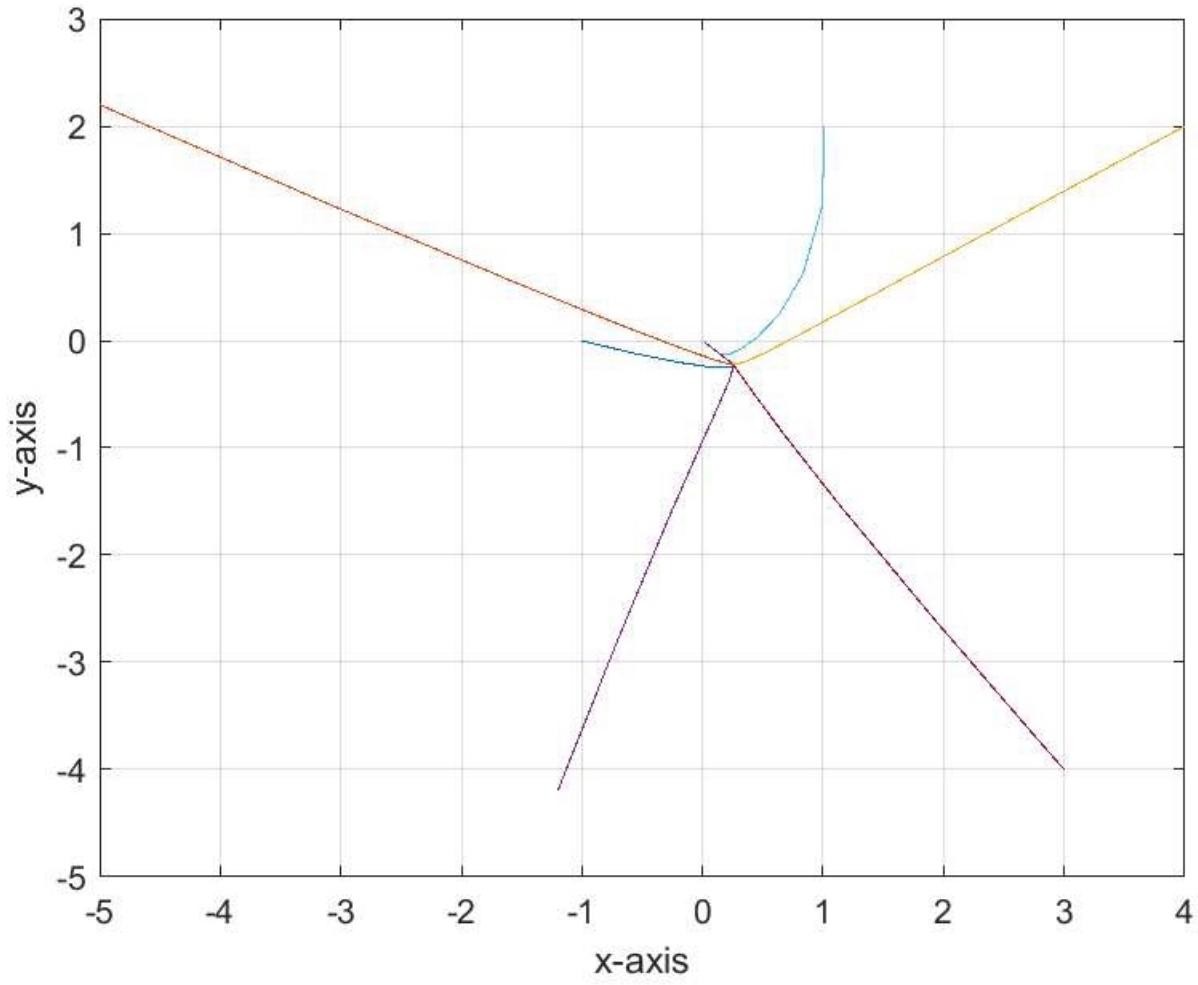


Figure 4.9: Output of the Six Robots' Trajectory on x-axis and y-axis, Reaching the Same Point.

This simulation shows that robots are capable of reaching consensus and forming certain shapes. The robots took more time to reach consensus than previous scenarios in Figure 4.2b and 4.4b since this graph in Figure 4.7 has less connections that the two mentioned. Which does not necessarily means a bad thing, the whole idea lies in the fact that robots just need a bit more time to communicate and make decisions.

Chapter 5

IMPLEMENTATION OF THE PROPOSED SYSTEM

Contents

The multi-robot system implementations that have been carried out are revealed in this chapter. Examining the framework configuration and the robots' building procedure in detail utilizing every technique described in the methodology chapter.

5.1 Experimental Setup

The setup of this project involves a framework where it served as the robots' environment. The robots are tested in a rectangular area measuring $\mathbf{2 \times 3m^2}$. A camera is mounted at a height of 2 meters, which provides a full view of the workspace. It is positioned centrally to ensure optimal coverage of the area, allowing precise robot tracking and detection, since it is essential for data collection. Figure 5.1 shows the setup.

Figure 5.1: Framework/ Environment of the Robots

5.2 Hardware Adjustments and Customizations

During the implementation phase of each robot, several modifications were required to ensure compatibility between components, particularly between the motor driver shield and the nRF communication module. The original pin configuration conflicted with the nRF communication module, so adjustments were required.

The original shield configuration used digital pins 11, 12, and 13, for more details on the shield's data sheet refer to the appendix 1 which conflicted with the SPI pins required by

the nRF module. To resolve this issue, both hardware and software modifications were implemented, ensuring that the driver shield could operate in parallel with the wireless module without interference. Figure 5.2 illustrates a comparison between the original and modified driver shield. The main changes are as follows:

Digital pins 11, 12 and 13 were originally soldered and connected, but these were removed in the modified version (highlighted in red). These pins are essential for the nrf module. Pins 11 and 12 were reassigned since they are both required for the nrf. Pin 10 was retained for the motor driver instead of 11, while pin 2 was used as a replacement of pin 12. These changes required both physical rewiring on the shield and corresponding updates to the driver library to reflect the new pin configuration; a modified version of the library was created to reflect the new pin configuration. This adjustment allowed the motor driver and the nRF module to coexist on the same microcontroller without any interference. The six robots were equipped with the modified shield to ensure consistent performance across the system.

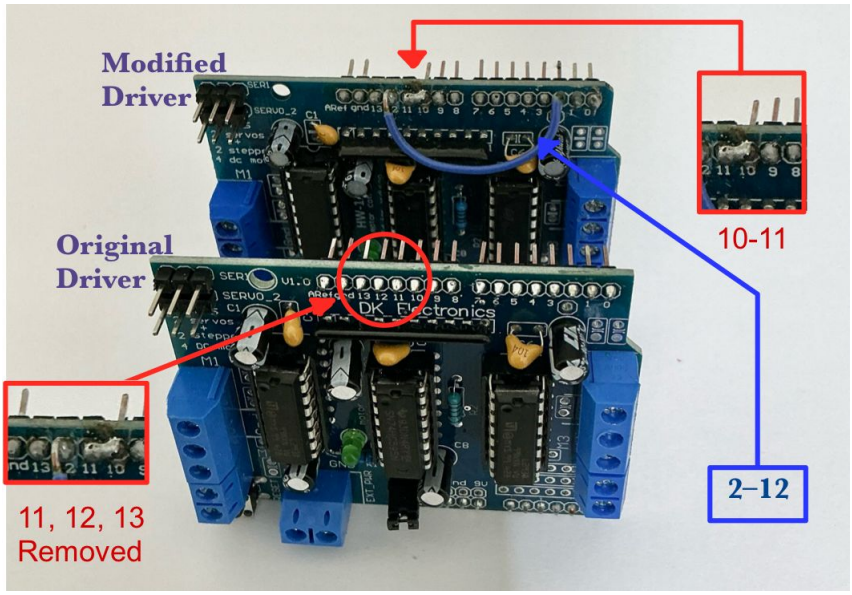


Figure 5.2: Customized Wiring on the Motor Driver Shield to Enable nRF24L01 Module Integration.

5.3 Wheels Configuration

Figure 5.3 illustrates the wheel configurations for the six robots implemented in the system. This configuration allows the robots to translate in any direction and rotate around their center, which is useful for coordinated motion and formation tasks.

Each wheel is assigned a specific index $W[i]$ representing its control input in the software, where $i = 0, 1, ..., 23$. For instance, let ω_1^1, ω_2^1, ω_3^1, and ω_4^1 are the motor speeds of the first robot, then we have the following mapping

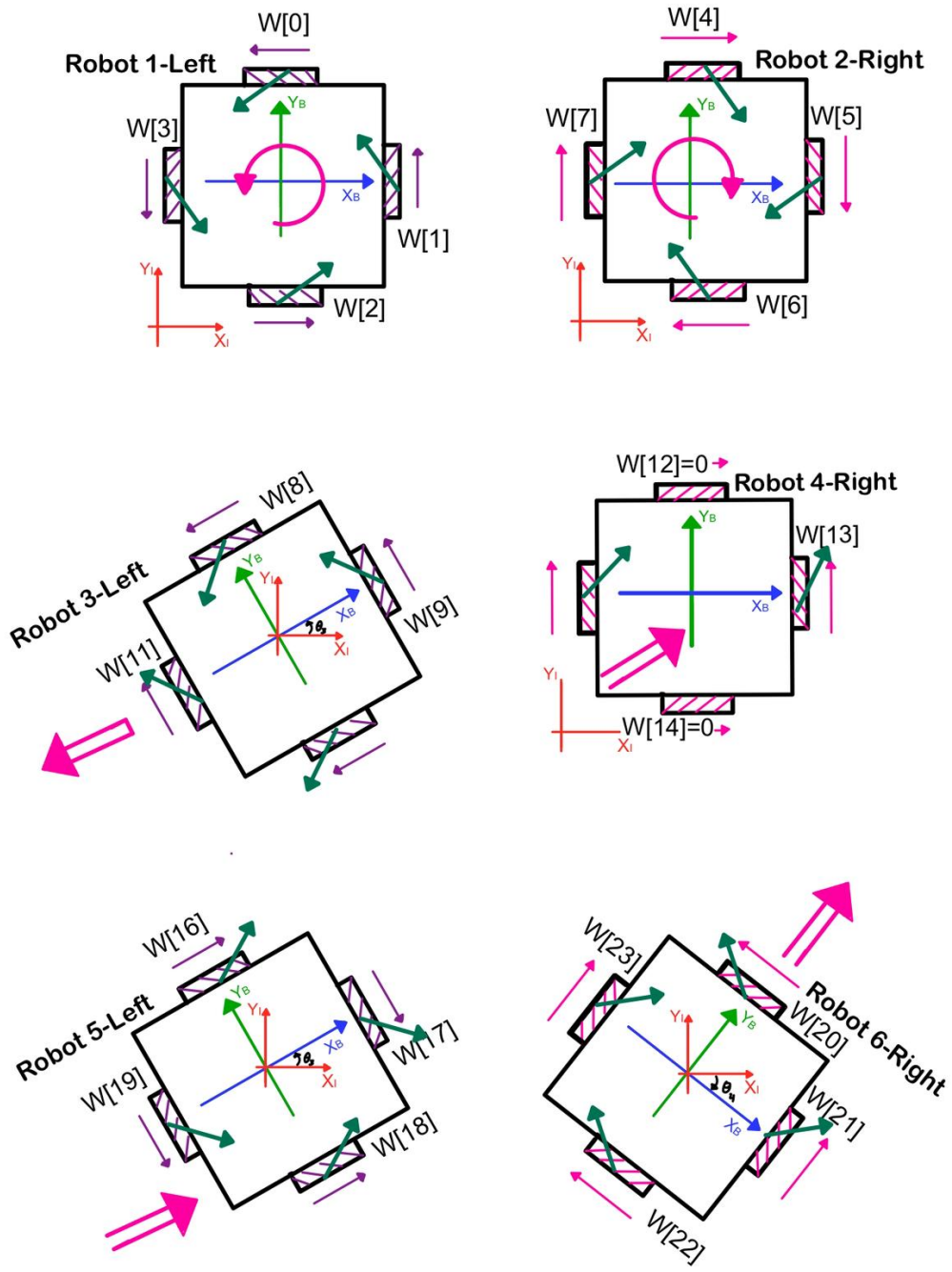
$$\begin{aligned} W[0] &\equiv \omega_1^1 \\ W[1] &\equiv \omega_2^1 \\ W[2] &\equiv \omega_3^1 \\ W[3] &\equiv \omega_4^1 \end{aligned}$$

and for the second robot, we have:

$$\begin{aligned} W[4] &\equiv \omega_1^2 \\ W[5] &\equiv \omega_2^2 \\ W[6] &\equiv \omega_3^2 \\ W[7] &\equiv \omega_4^2 \end{aligned}$$

and so on.

The pink arrows indicate the resultant motion direction, while the green and blue axes represent the body frame (X_B, Y_B), and red axes denote the inertial (global) frame (X, Y). Robots 1, 3, and 5 use left Mecanum wheels, while robots 2, 4, and 6 use right Mecanum wheels. This configuration ensures unified control and flexible shape reconfiguration across the multi-robot system.



]

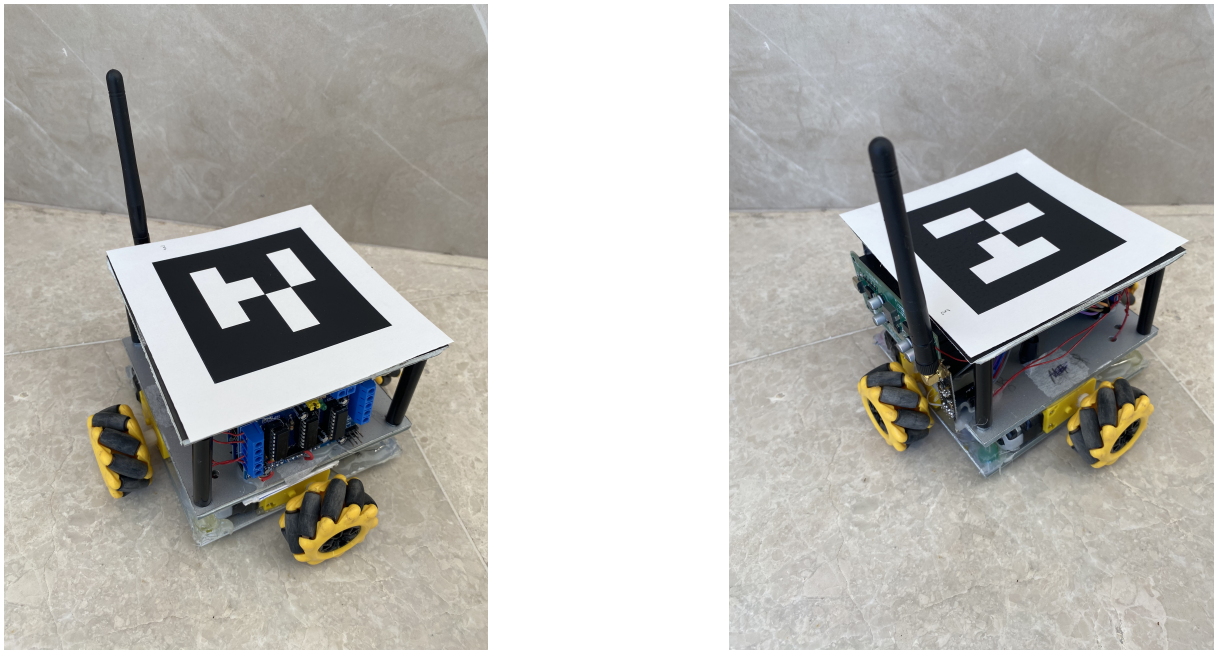
Figure 5.3: Wheel configurations and examples illustrating the motion of robots in the inertial and body coordinate systems.

5.4 Robots Implementation

The hardware implementation of the robots was carried out according to the plan outlined in chapter 3. Each robot was assembled and designed to ensure that all components fit

together seamlessly and function as desired. The first phase in the process of assembling the robot was designing and building the base, which serves as the body of the robot. This solid foundation supports the motors to which the mecanum wheels are attached, allowing the robot to move smoothly, as stated above. For the purpose of maximizing space, maintaining balance, and keeping everything within reach in case of any modifications or battery charging, all components were placed on the sides as shown in Figure 5.4.

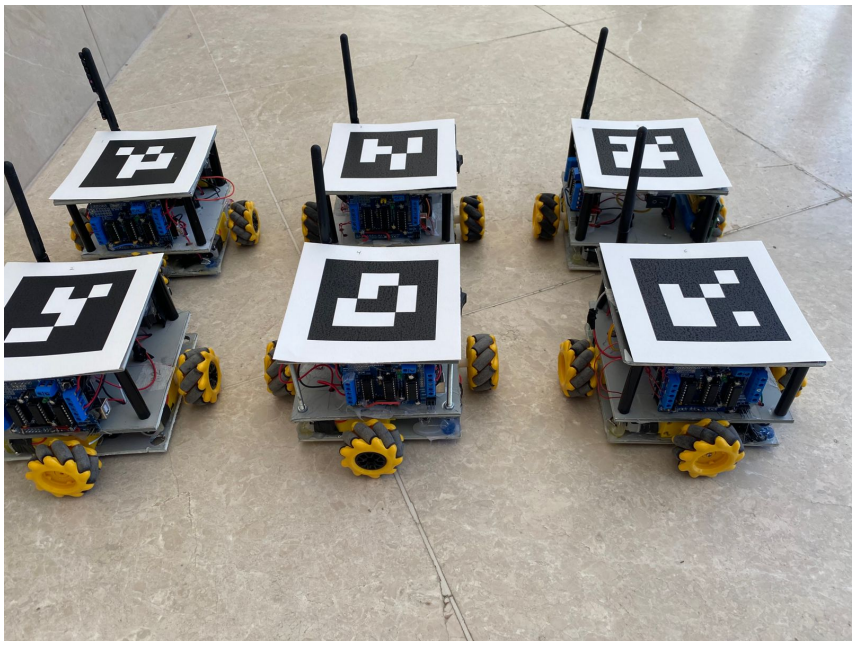
Lastly, to ensure that the robots could be easily detected and tracked, a top plate was added to hold the AprilTag, which is placed so it is visible to the camera system. The placement was important to ensure accurate localization. The same assembly process was repeated for all six robots, ensuring they were identical in functionality.



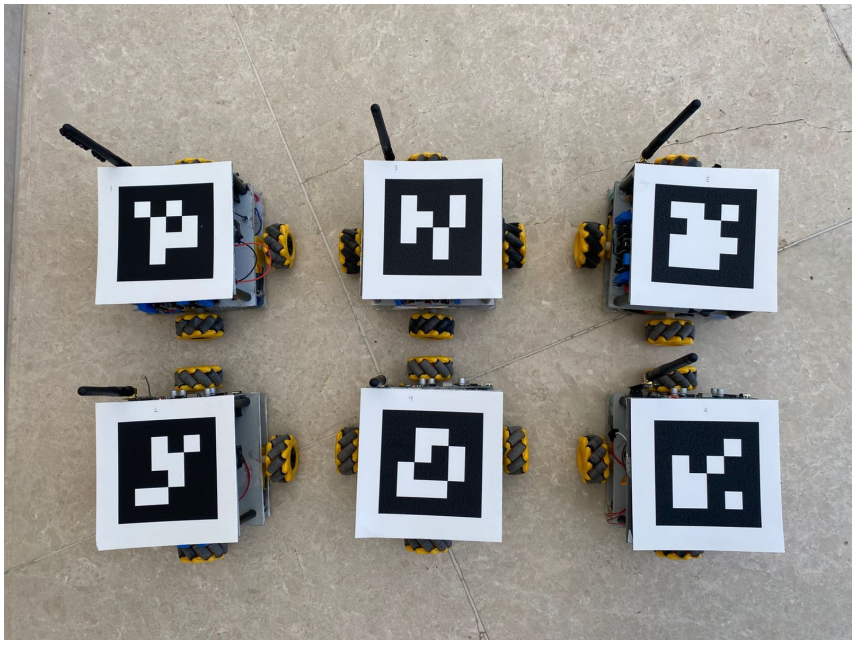
(a) Front View of the Robot (b) Side View of the Robot

Figure 5.4: Multiple Views of the Robot with all its Components

After completing the assembly process described before, six identical robots were built. Figure 5.5 shows the six fully assembled robots. Each robot was tested to ensure it functioned properly before being integrated into the multi robot system.



(a) Front view of the robots



(b) Top view of the robots

Figure 5.5: The six assembled robots used in the system

Chapter 6

EXPERIMENTAL RESULTS

Contents

6.1 Local Positioning System

To test the positioning system, 17 AprilTags were used without any preprocessing. The following figures show the LPS results without preprocessing the frames. The system detects 5 robots/tags that do not exist within the camera's field of view. This happens due to the extreme noise present in each frame. It is observed that 42 non existing AprilTags were detected within one minute.

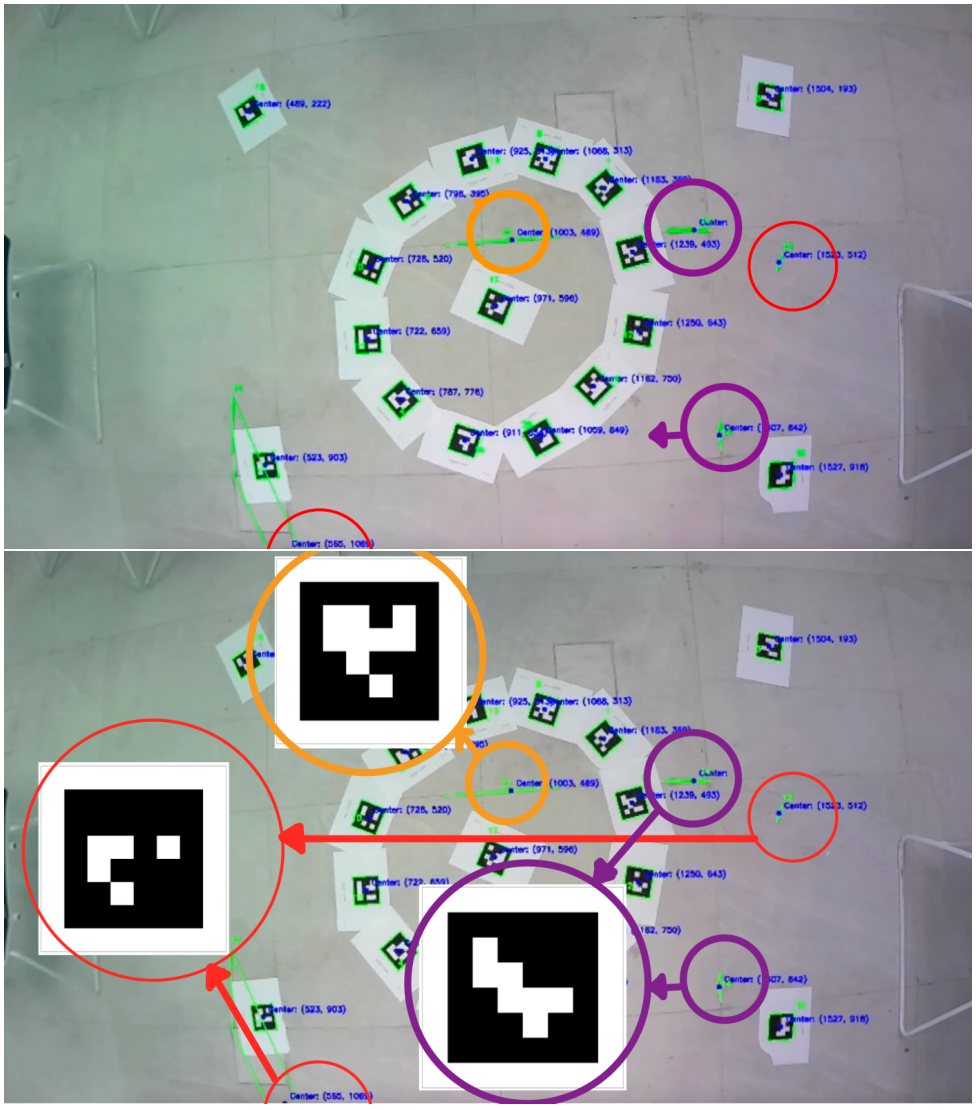
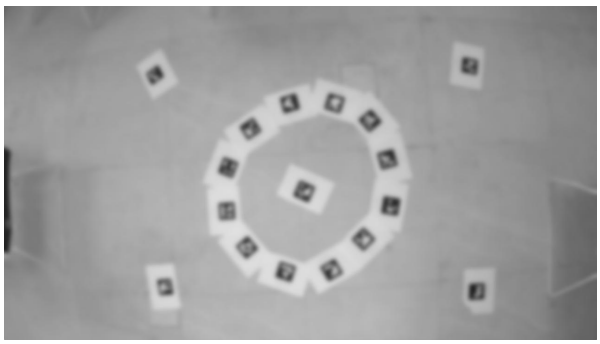
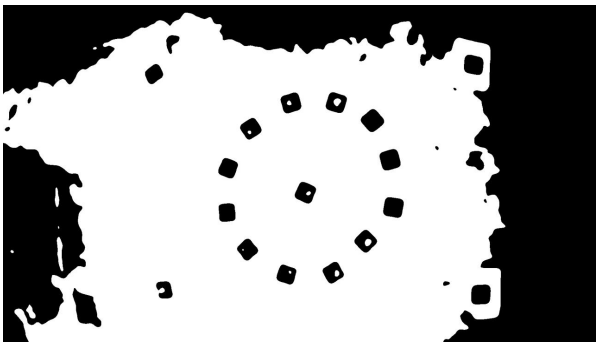


Figure 6.1: Captures without Preprocessing the Frames.

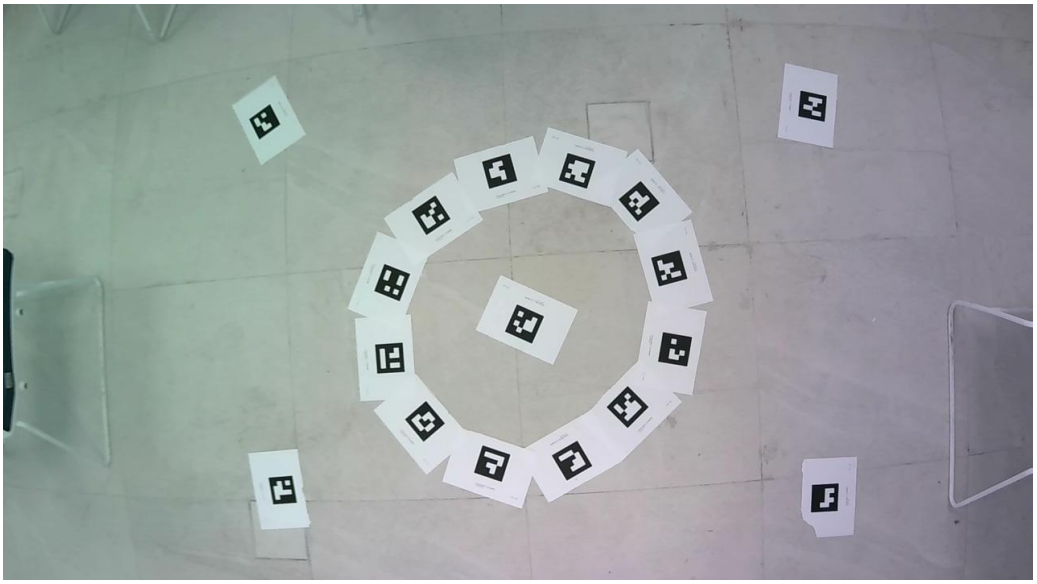
To overcome this problem, a Gaussian blurring was applied and converted the images to binary format. Figure 6.2 shows the results of this preprocessing using a blur radius of 45 pixels. Figure 6.2a illustrates the gray image after blurring, Figure 6.2b shows the binary image generated after blurring. The blurring process is too intense, preventing the detection of any AprilTag. While it eliminates false positives, it introduces the issue of false negatives.



(a) Gray Scale with 45 Pixel Blurring.



(b) Binary Photo with 150 Threshold.



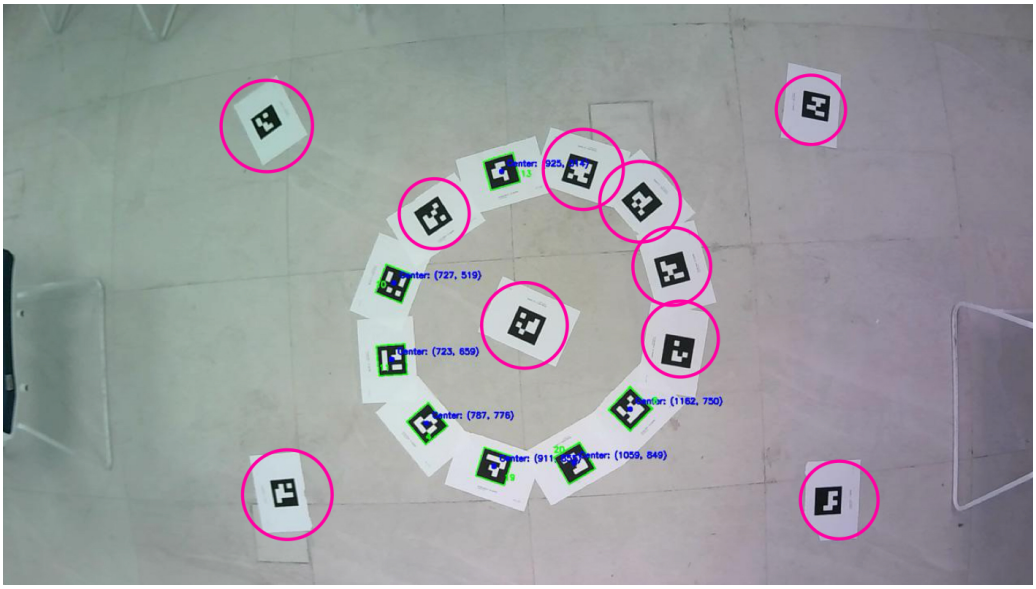
(c) The Capture with no Detected Tags.

Figure 6.2: Captures with 150 Threshold and 45 Pixel Blurring.

The previous results lead us to reduce the blurring of the gray image. Figure 6.3 shows the gray, binary, and the original image for 150 threshold and 25 pexils of blurring. It is observed that 10 robots are not detected out of 17 robots. Moreover, the false positive does not occur while the false negative still exists. This means that one can compromise between the no-blurring and the intense blurring.

(a) Gray Scale with 25 Pixel Blurring.

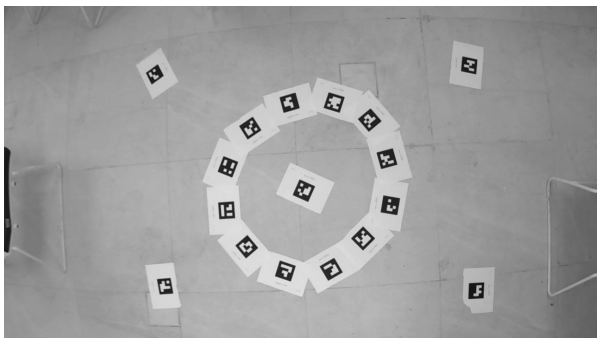
(b) Binary Photo with 150 Threshold.



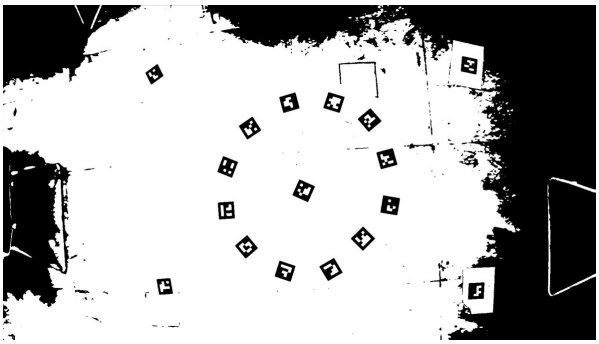
(c) Only 7 Robots Out of 17 Detected.

Figure 6.3: Captures with 150-Threshold and 25-Pixel Blurring.

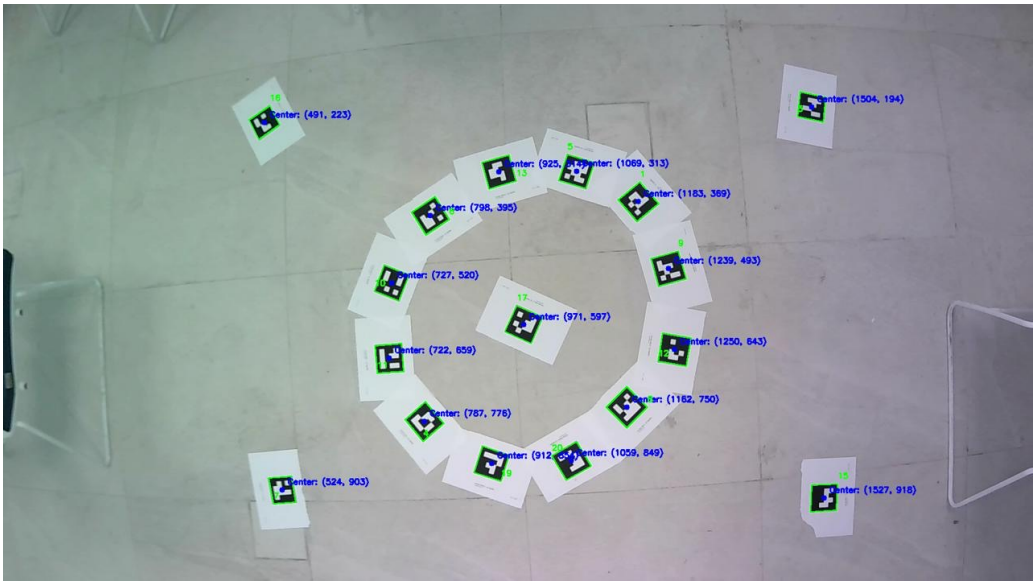
When using 5 pixel blurring, all tags were detected appropriately and no false positive occurred. These results are not good enough, since there exists noise in the binary image as shown in Figure 6.4b.



(a) Gray scale with 5 Pixel Blurring.



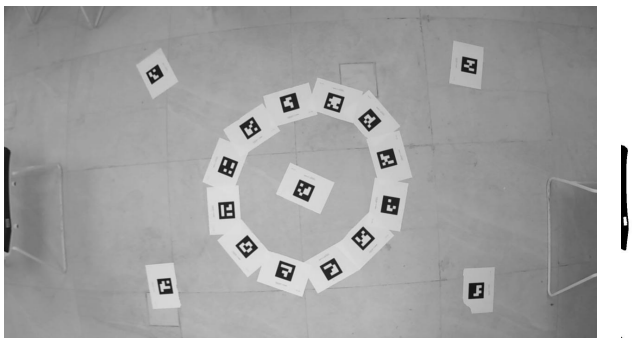
(b) Binary Photo with 150 Threshold.



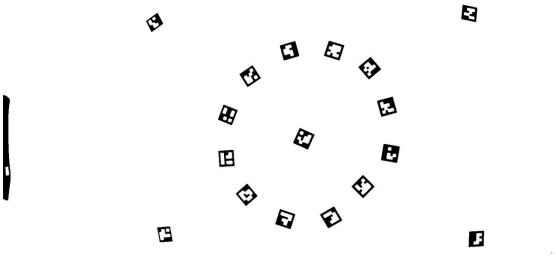
(c) 17 Robots out of 17 Detected .

Figure 6.4: Captures with 150 Threshold and 5 Pixel Blurring.

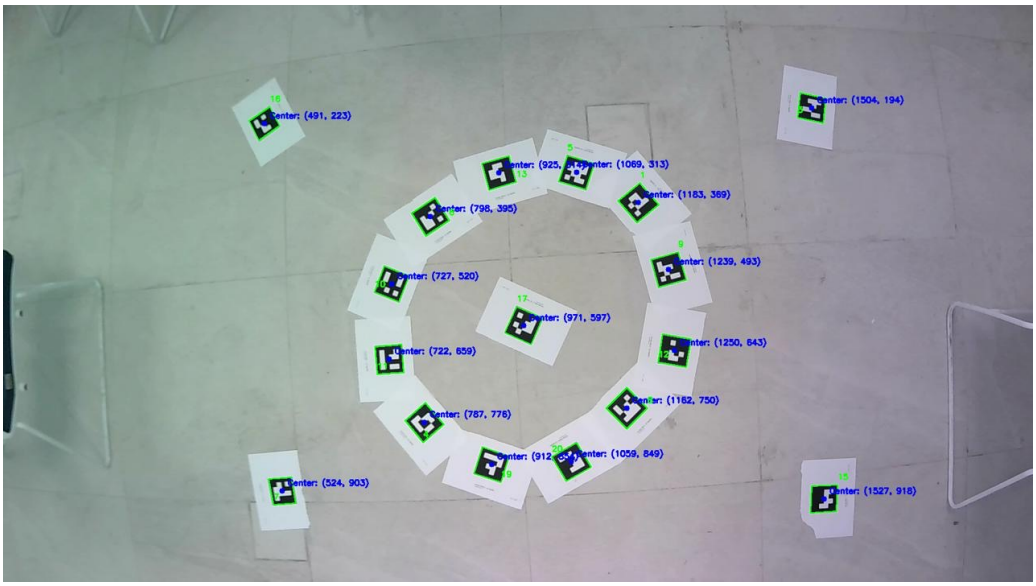
The last results are appropriate since no false postives and no false negatives occur. However, we want to refine the binary image to remove all the noise. A threshold with a value of 50 is used. The results are shown in Figure 6.5c. All the tags were detected accurately, and no false positives occurred.



(a) Gray Scale with 5 Pixel Blurring.

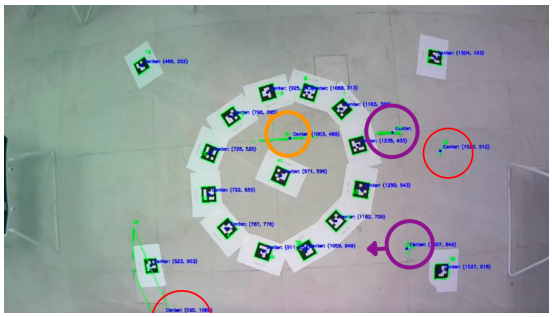


(b) Binary Photo with 50 Threshold.

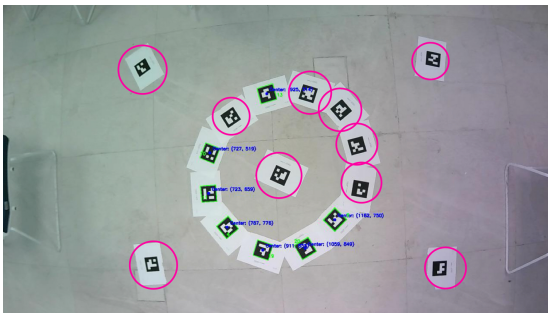


(c) All Robots were Detected.

Figure 6.5: Captures with 50-Threshold and 5-Pixel Blurring.

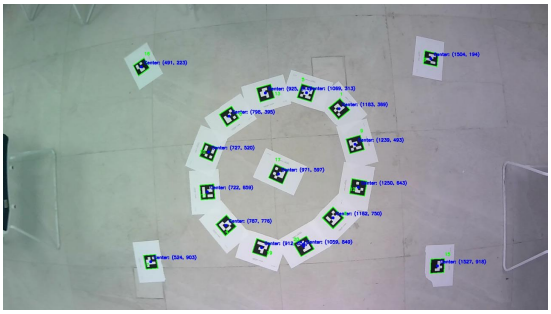


(a) False Positive Case: Non-existing Tags have been Detected due to Noise (No blurring).



(b) False Negative Case: Only 7 Robots Out of 17 Detected due to Moderate Blurring.

(c) Extreme False Negative Case: No Detection at all Due to an Intense Blurring



(d) True Positive: All Robots have been Detected with Appropriate Blurring and Binary Threshold.

Figure 6.6: Positioning System Results for all Cases.

6.2 Performance Evaluation of Single Robot Control

The experimental evaluation of the single robot control was carried out using a camera-based localization system integrated with AprilTag markers, enabling precise determination of the robot's position and orientation within the workspace. The robot's position and orientation were continuously monitored by the vision system, providing real-time feedback for the control loop.

Initially, the robot was controlled without a PID controller, relying solely on open-loop commands generated from the localization feedback. This approach resulted in a significant steady-state error in both position and orientation, highlighting an inadequate capacity to correct persistent deviations from the desired trajectory. Proportional control

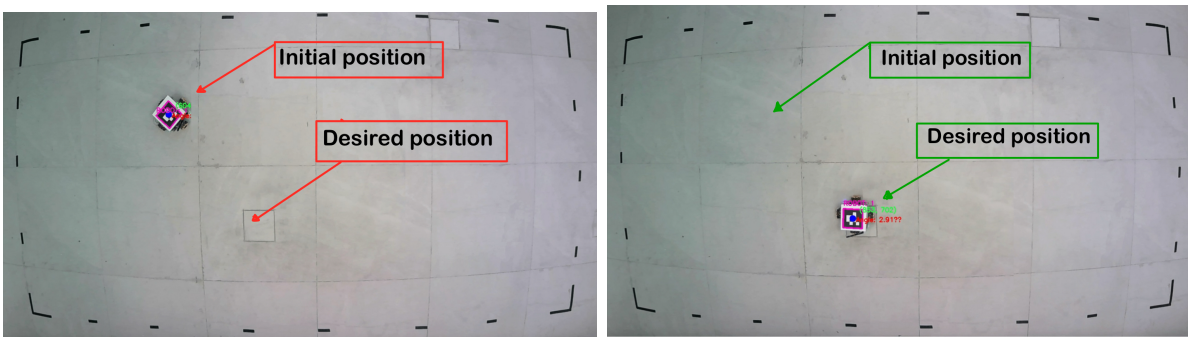
gains were subsequently introduced and incrementally increased in an effort to minimize the steady-state error. However, high gain values induced oscillatory behavior and pronounced overshoot, compromising the stability and accuracy of the robot's motion. The PID controller implemented in this experiment followed the design outlined in the Methodology section. In the absence of the PID controller, steady-state errors persisted, and increasing the proportional gain alone led to oscillations and overshoot. Incorporating the full PID structure-comprising proportional, integral, and derivative terms-eliminated the steady-state error and effectively suppressed oscillations, resulting in stable and precise control of the robot's position and orientation. The process of PID tuning involved testing various gain configurations to evaluate their effects on the system's performance. Among these configurations, the best performance was achieved when implementing reset control for the integral term. This reset control effectively prevented integral windup, ensuring faster convergence to the desired trajectory while maintaining system stability. These experimental results demonstrate the progression of control quality across different gain configurations and underscore the importance of integrating all three PID components-particularly the implementation of reset control for the integral term to achieve optimal performance.

Table 6.1: Effect of different PID gains on single robot control

K_p	K_i	K_d	**Configuration**	**Observed Behavior**
2	0	0	P-only	Persistent steady-state error, slow convergence
10	0	0	High P-only	Fast response, significant overshoot, oscillations
5	0	0	Moderate P-only	Smaller steady-state error, mild oscillations
5	0	1	PD control	Reduced oscillations, steady-state error still present
0	0.5	0	I-only	Eliminated steady-state error, slow response, risk of windup
1	0.5	0	PI control	Reduced steady-state error, some overshoot, moderate stability
1.5	0.6	0.1	Full PID	No steady-state error, moderate overshoot, stable
0.5	0.6	0.3	Full PID	No steady-state error, minimal overshoot, stable, accurate tracking
0.5	**0.6**	**0.3**	**PID - Reset**	**No steady-state error, No overshoot, stable, accurate tracking**

Figure 6.7a shows the robot at its starting position, located at $x = 604, y = 347$, with an angle of $\theta = -50$. Figure 6.7b shows the robot after reaching its target position at $x = 850$, $y = 700$ with an angle of $\theta = 0$. The robot moved from the initial to the final position in 3 seconds using a PID controller with a reset mechanism for the integral term. This controller helped reduce steady-state errors and prevented the build-up of error over time. As a result, the robot followed the desired path accurately and reached the final

position smoothly and stably.



(a) Initial position t=0s (b) Final position at t=3s

Figure 6.7: Single Robot Controlled by a PID-Reset Controller

6.3 Validation of Coordinated Motion of Multiple Robots

Building on the experimental results from single-robot control, the control methodology was expanded to coordinate the motion of six robots. Each robot operated with its own local control loop, using the tuned PID gains to ensure accurate trajectory tracking and stable performance during cooperative tasks. To enable coordinated movement among the robots, different interaction topologies were applied. These were represented by different Laplacian matrices, each defining specific communication and influence patterns within the group. By using various Laplacian matrices, the system was tested for its ability to maintain formation geometry and adapt to different interaction structures.

6.3.1 First Experiment (Fully Connected Topology with Uniform Interaction)

In the first experiment, the multi-robot system was tested using a fully connected interaction topology characterized by the Laplacian matrix L:

$$\mathcal{L} = \begin{bmatrix} 5 & -1 & -1 & -1 & -1 & -1 \\ -1 & 5 & -1 & -1 & -1 & -1 \\ -1 & -1 & 5 & -1 & -1 & -1 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & -1 & 5 & -1 \\ -1 & -1 & -1 & -1 & -1 & 5 \end{bmatrix}$$

This matrix represents a fully connected graph where each robot influences every other robot equally. The formation of the six robots in the workspace were set as follows:

$$\text{Robot 1: } (r_{x1}, r_{y1}, r_{z1}) = (500, 500, 0)$$

$$\text{Robot 2: } (r_{x2}, r_{y2}, r_{z2}) = (700, 500, 0)$$

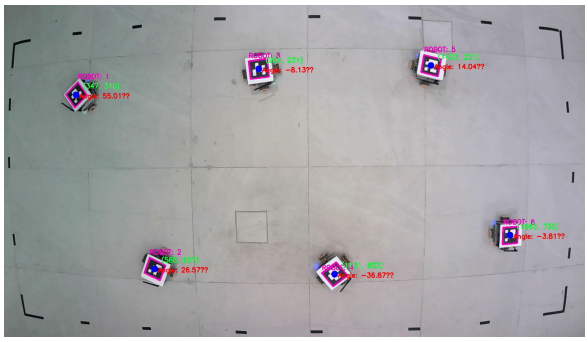
$$\text{Robot 3: } (r_{x3}, r_{y3}, r_{z3}) = (900, 500, 0)$$

$$\text{Robot 4: } (r_{x4}, r_{y4}, r_{z4}) = (1100, 500, 0)$$

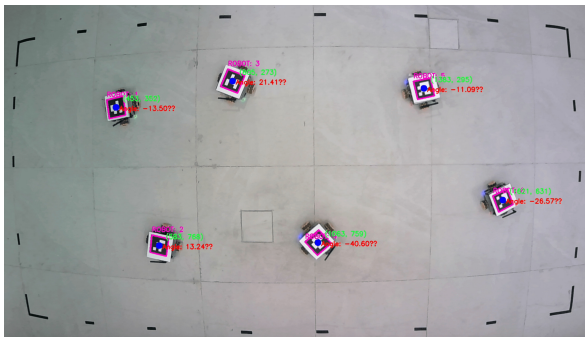
$$\text{Robot 5: } (r_{x5}, r_{y5}, r_{z5}) = (1300, 500, 0)$$

$$\text{Robot 6: } (r_{x6}, r_{y6}, r_{z6}) = (1500, 500, 0)$$

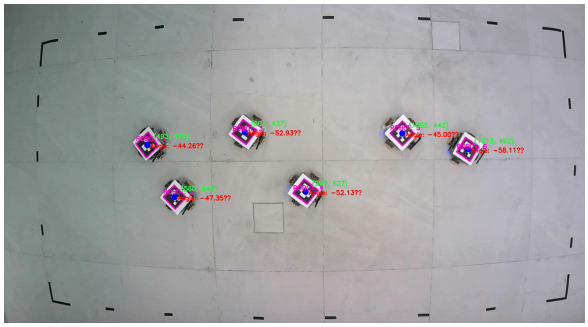
This configuration placed the robots in a line formation within the workspace, where each robot maintains equal distance of 200 pixels on the x-axis from its neighbors in a straight alignment. Figure 6.8 illustrates the resulting spatial formation. All six robots were able to maintain their relative positions and orientation in nearly 8 seconds, demonstrating successful coordination under the complete graph model.



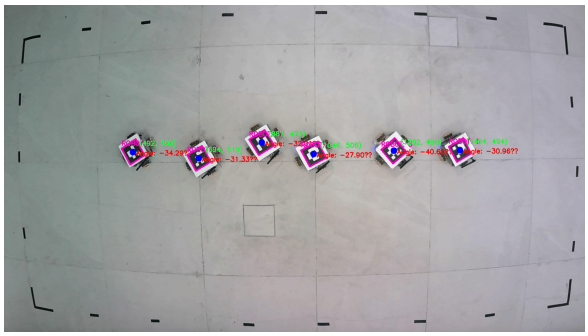
(a) Initial position t=0s



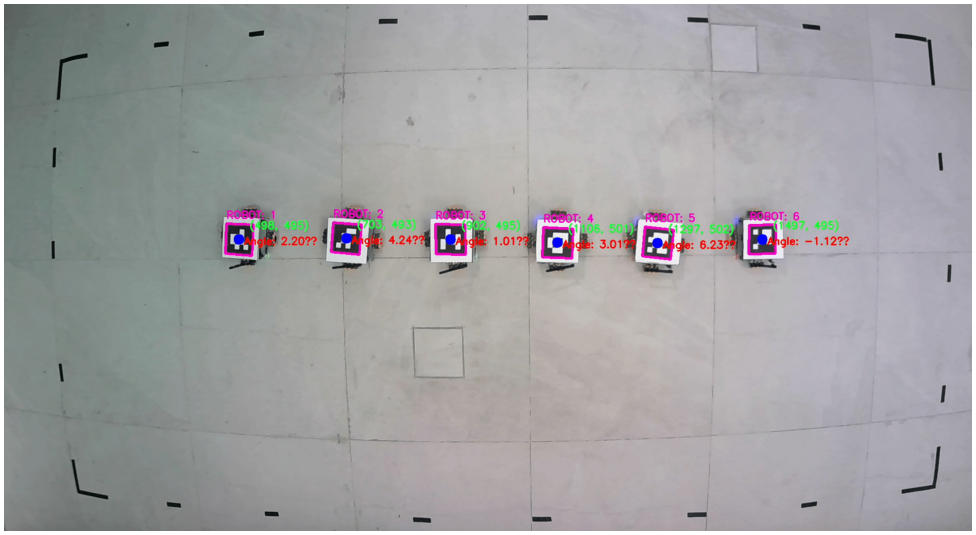
(b) t=2s



(c) t=4s



(d) t=6s



(e) Final position at t=8s

Figure 6.8: Formation process of a line by 6 robots over 8 seconds

During experimentation, each robot's individual motion was governed by separate PID loops, which adjusted actuator commands based on the error between desired and actual states which are the position and orientation.The proportional component corrected immediate errors, the integral component handled steady-state errors, and the derivative component predicted future errors to reduce overshoot and improve stability. The PID

parameters were tuned empirically to optimize the response for each robot under different conditions. Results demonstrated that PID control effectively minimized errors and ensured smooth movement. Overall, the experimental results confirm that PID control is a simple yet effective solution for managing multi-robot systems in real time.

6.3.2 Second Experiment (Partially Connected Graph)

In this phase, an implementation of a decentralized formation control system that was proposed and simulated in Chapter 4. To evaluate its performance in a real-world setting, this topology was tested with two different formations: a rectangle and a triangle. These formations were chosen to demonstrate the system's ability to coordinate multiple agents into both structured and compact shapes using only local interactions and decentralized control. The topology of communication among the robots is represented by this Laplacian matrix:

$$\mathcal{L} = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}$$

6.3.2.1 Rectangle Formation

For the rectangle formation, the robots are placed in the workspace with the following coordinates:

$$\text{Robot 1: } (r_{x1}, r_{y1}, r_{z1}) = (300, 500, 0)$$

$$\text{Robot 2: } (r_{x2}, r_{y2}, r_{z2}) = (300, 200, 0)$$

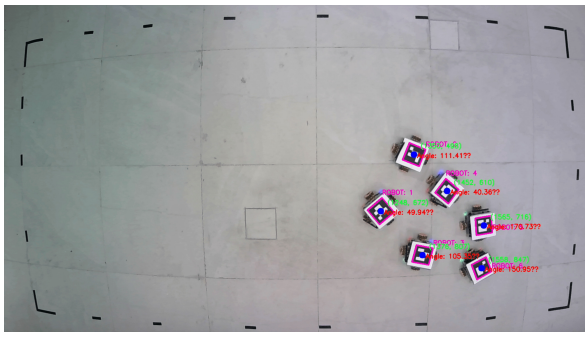
$$\text{Robot 3: } (r_{x3}, r_{y3}, r_{z3}) = (600, 500, 0)$$

$$\text{Robot 4: } (r_{x4}, r_{y4}, r_{z4}) = (600, 200, 0)$$

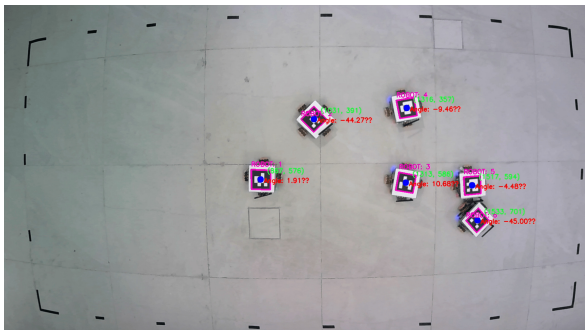
$$\text{Robot 5: } (r_{x5}, r_{y5}, r_{z5}) = (900, 500, 0)$$

$$\text{Robot 6: } (r_{x6}, r_{y6}, r_{z6}) = (900, 200, 0)$$

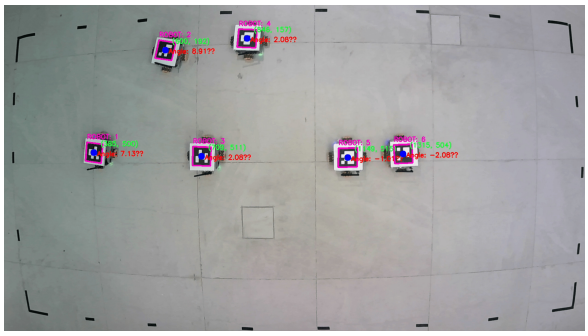
This configuration placed the robots in a rectangle formation within the workspace. Figure 6.9 illustrates the resulting formation. All six robots were able to maintain their positions and orientation in 10 seconds, demonstrating successful coordination.



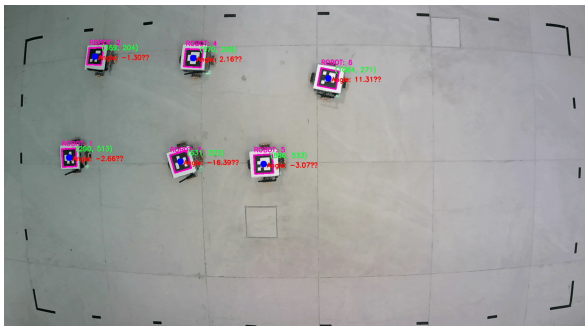
(a) Initial position t=0s



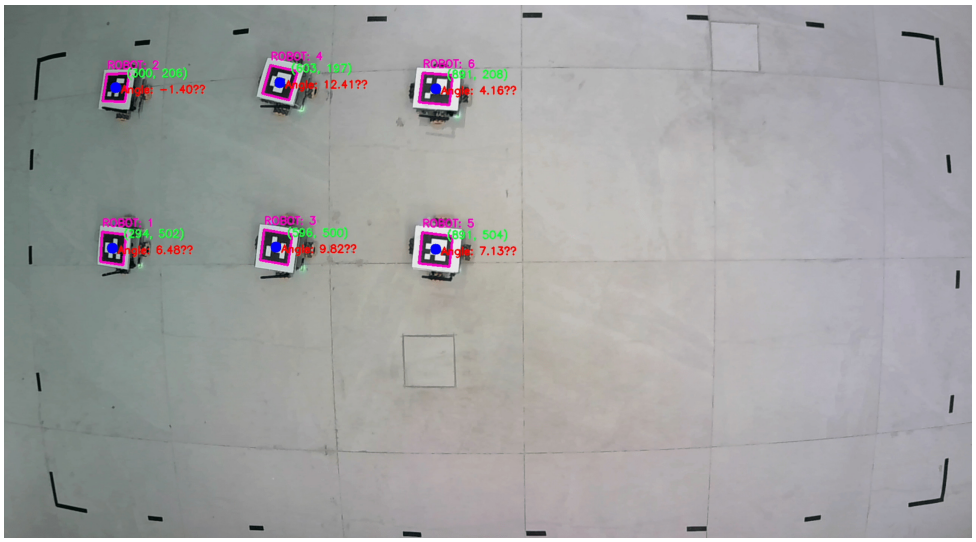
(b) t=2s



(c) t=6s



(d) t=8s



(e) Final position at t=10s

Figure 6.9: Formation process of a rectangle by 6 robots over 10 seconds

6.3.2.2 Triangle Formation

For the triangle formation, implementing the same Laplacian matrix, the robots were positioned in the workspace with the following coordinates:

$$\text{Robot 1: } (r_{x1}, r_{y1}, r_{z1}) = (300, 500, 0)$$

$$\text{Robot 2: } (r_{x2}, r_{y2}, r_{z2}) = (300, 200, 0)$$

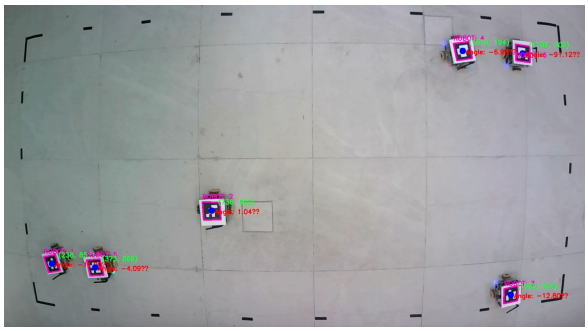
$$\text{Robot 3: } (r_{x3}, r_{y3}, r_{z3}) = (600, 500, 0)$$

$$\text{Robot 4: } (r_{x4}, r_{y4}, r_{z4}) = (600, 200, 0)$$

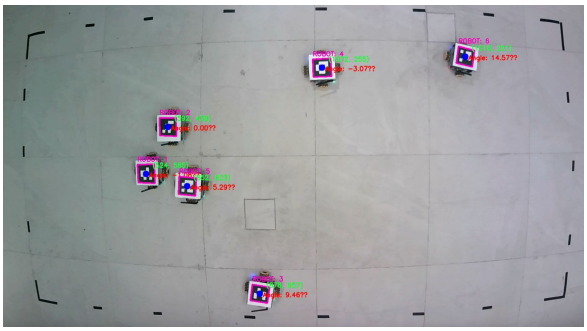
$$\text{Robot 5: } (r_{x5}, r_{y5}, r_{z5}) = (900, 500, 0)$$

$$\text{Robot 6: } (r_{x6}, r_{y6}, r_{z6}) = (900, 200, 0)$$

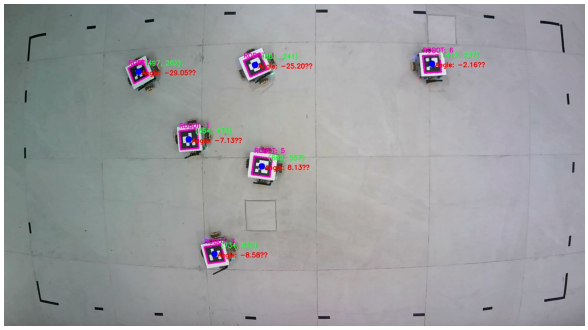
This configuration arranged the robots into a triangular structure within the workspace. Figure 6.10 illustrates the resulting formation. All six robots successfully reached and maintained their designated positions within 7 seconds, confirming the effectiveness of the coordination algorithm in forming a structured triangular shape.



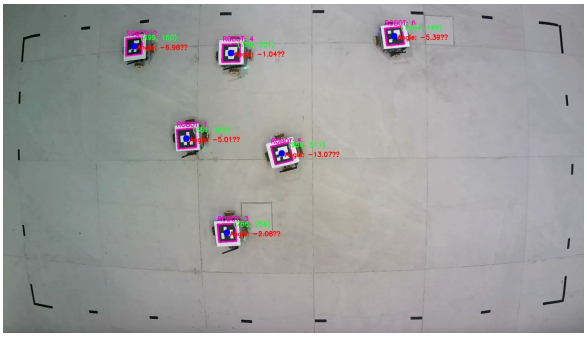
(a) Initial position t=0s



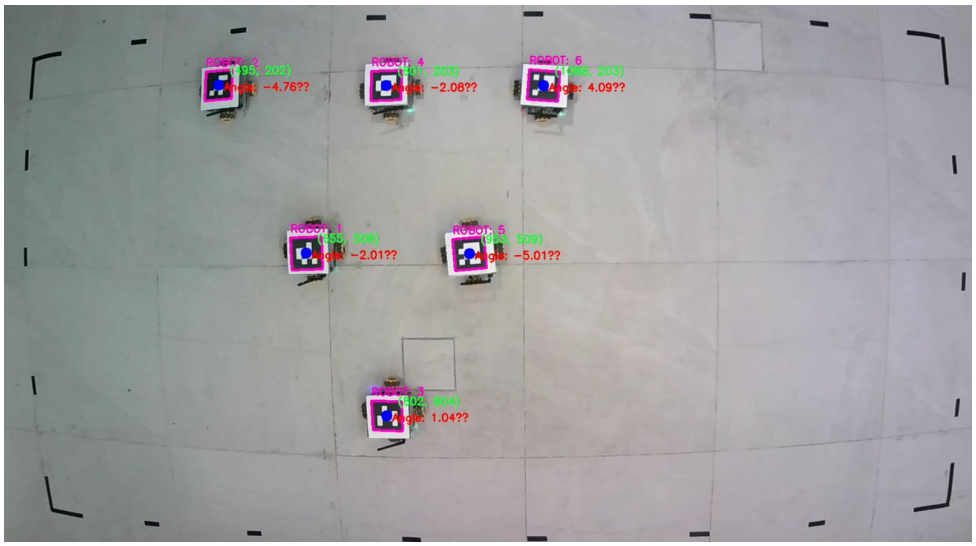
(b) t=3s



(c) t=5s



(d) t=6s



(e) Final position at t=7s

Figure 6.10: Formation process of a triangle by 6 robots over 7 seconds

6.3.3 Third Experiment: Leader-Follower Topology

In this setup, a leader-follower topology was implemented, where one robot acts as the leader and the remaining five robots operate as followers. The followers receive information only from the leader, with no communication occurring among themselves. This

configuration reduces communication complexity and emphasizes the impact of the leader on the overall formation. The Laplacian matrix for this topology is represented:

$$\mathcal{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Two triangular formations were tested under this topology. In the first trial, the leader was placed at at a certain point, guiding the other robots to form a stable triangular structure behind it. In the second trial, the leader's position was shifted to a different coordinates of the workspace. The followers dynamically adjusted their positions based on the change of leader's location, reforming the triangle around the updated reference. These experiments demonstrate the flexibility and responsiveness of the multi-robot system under a leader-based control strategy. Coordination for the first triangle formation is the following:

$$\text{Robot 1: } (r_{x1}, r_{y1}, r_{z1}) = (650, 500, 0)$$

$$\text{Robot 2: } (r_{x2}, r_{y2}, r_{z2}) = (500, 200, 0)$$

$$\text{Robot 3: } (r_{x3}, r_{y3}, r_{z3}) = (800, 800, 0)$$

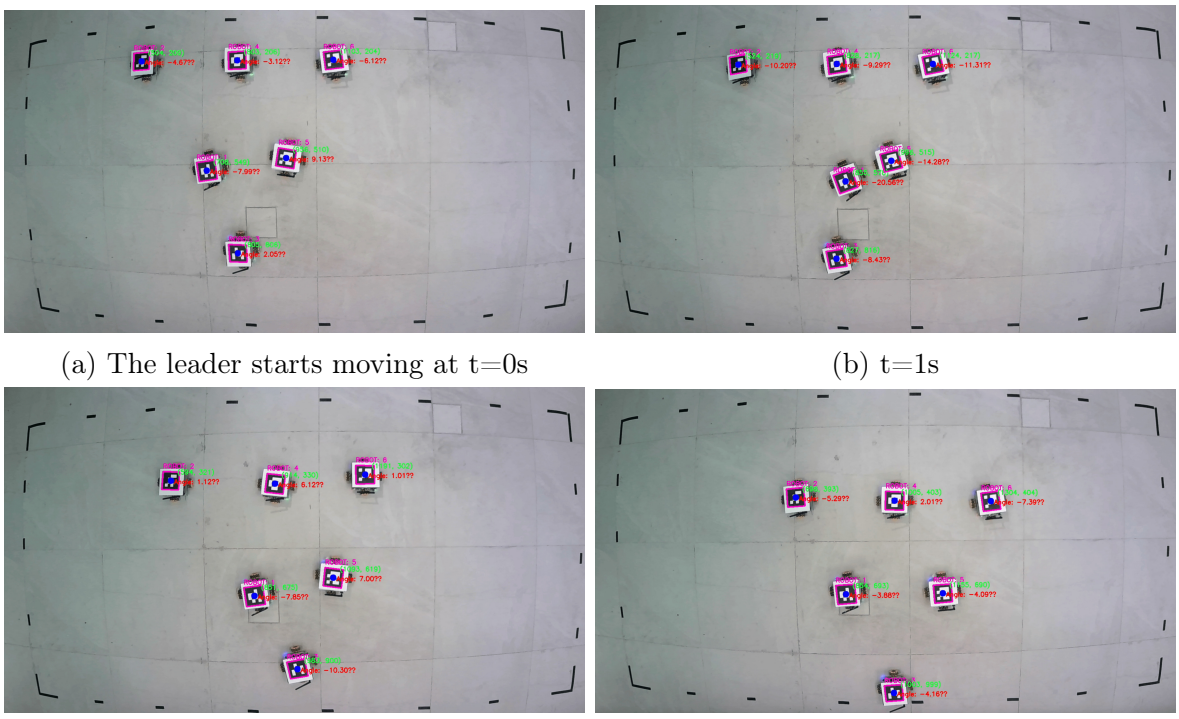
$$\text{Robot 4: } (r_{x4}, r_{y4}, r_{z4}) = (800, 200, 0)$$

$$\text{Robot 5: } (r_{x5}, r_{y5}, r_{z5}) = (950, 500, 0)$$

$$\text{Robot 6: } (r_{x6}, r_{y6}, r_{z6}) = (1100, 200, 0)$$

The formation of the triangle was successful in under 7s. After this was achieved, the position of the leader was changed to this coordinates: Robot 1: $(r_{x1}, r_{y1}, r_{z1}) = (850, 700, 0)$

and all robots adjusted themselves to shape the triangle in the new location of the leader. Figure 6.11 shows the formation of the triangle in the new location.



(a) The leader starts moving at t=0s (b) t=1s

(c) Followers changing their positions t=3s (d) Final coordination of the followers at t=5s

Figure 6.11: Leader following process of 5 followers over 5 seconds

In both cases, the formation was successfully achieved within 7 seconds for the first triangle, and 5s for the second. Confirming the system's ability to maintain structure and stability with a minimal communication topology.

Chapter 7

CONCLUSIONS AND FUTURE WORK

Contents

7.1 Conclusions

This project has demonstrated how several robots can collaborate effectively, combining modern technology with strategic design to achieve coordination, consensus, and formation. These robots are an example of what can be accomplished when innovative tools are combined. Every robot has nRF modules for wireless connection and mecanum wheels for smooth movement. Also, a localization system to detect and localize robots in the framework. The functionality, reliability, and efficiency of the system have been proven

through multiple simulations, where robots have demonstrated their ability to collaborate, reach consensus, and form shapes. Lastly, this project highlights the potential of the multi-robot system laying the groundwork for future advancements and more exciting possibilities, such as smarter decision making and achieving more complex challenges.

7.2 Future Work

Adding more to the project, a promising direction for future work is to enhance the functionality of the robotic system by introducing defense and protection mechanisms for a leader robot. This improvement requires implementing algorithms that allow robots to collaborate and work together to make autonomous decisions in real time to accomplish their goal of removing danger and safeguarding their leader. The system can be improved by creating protective formations around the leader, which will allow the robots to arrange themselves to offer coverage and reduce exposure to possible threats. Their duties include both defending and safeguarding the leader. An advanced threat detection and response mechanism would allow robots to identify and address threats and danger effectively, where they work as an army. These developments would make the project a more advanced multi-robot system that can defend itself cooperatively.

In addition, another valuable enhancement would be adding collision avoidance strategies. Since multiple robots are moving and operating together in the same environment, preventing them from colliding is essential to ensure smooth and safe operation. Future work could explore real-time obstacle detection and smart avoidance algorithms that help each robot adjust its path on the go, to allow robots to adapt their paths dynamically while maintaining formation and coordination.

References

[1] G. S. Seyboth, W. Ren, and F. Allgöwer, "Cooperative control of linear multi-agent systems via distributed output regulation and transient synchronization," *Automatica*, vol. 68, pp. 132–139, 2016.

[2] F. Chen, W. Ren, *et al.*, "On the control of multi-agent systems: A survey," *Foundations and Trends® in Systems and Control*, vol. 6, no. 4, pp. 339–499, 2019.

[3] T. Arai, "Collision avoidance among multiple robots using virtual impedance," in *Proc. IEEE/RSJ Int. Workshop Intel. Robots and Systems' 89*, 1989.

[4] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert, "Multi-robot cooperation in the martha project," *IEEE Robotics & Automation Magazine*, vol. 5, no. 1, pp. 36–47, 1998.

[5] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE transactions on robotics and automation*, vol. 14, no. 6, pp. 926–939, 1998.

[6] S. A. DeLoach, E. T. Matson, and Y. Li, "Applying agent oriented software engineering to cooperative robotics.," in *FLAIRS*, pp. 391–396, 2002.

[7] C. G. Cena, P. F. Cardenas, R. S. Pazmino, L. Puglisi, and R. A. Santonja, "A cooperative multi-agent robotics system: Design and modelling," *Expert Systems with Applications*, vol. 40, no. 12, pp. 4737–4748, 2013.

[8] G. Dudek, M. R. Jenkin, E. Milios, and D. Wilkes, "A taxonomy for multi-agent robotics," *Autonomous Robots*, vol. 3, pp. 375–397, 1996.

[9] N. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[10] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28573–28593, 2018.

[11] L. Ma, H. Min, S. Wang, Y. Liu, and S. Liao, "An overview of research in distributed attitude coordination control," *IEEE/CAA Journal of Automatica Sinica*, vol. 2, no. 2, pp. 121–133, 2015.

[12] A. Salvador Palau, M. H. Dhada, and A. K. Parlikad, "Multi-agent system architectures for collaborative prognostics," *Journal of Intelligent Manufacturing*, vol. 30, no. 8, pp. 2999–3013, 2019.

[13] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *International conference on machine learning*, pp. 5872–5881, PMLR, 2018.

[14] M. S. Mahmoud and B. J. Karaki, "Output-synchronization of discrete-time multi-agent systems: A cooperative event-triggered dissipative approach," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 114–125, 2020.

[15] T. Eren, P. Belhumeur, and A. Morse, "Coordination of groups of mobile agents using nearest neighbor rules," in *Proc. of the IEEE Conference on Decision and Control*, 2002.

[16] R. R. Negenborn, B. De Schutter, and J. Hellendoorn, "Multi-agent model predictive control: A survey," *arXiv preprint arXiv:0908.1076*, 2009.

[17] M. Ji and M. Egerstedt, "Distributed coordination control of multiagent systems while preserving connectedness," *IEEE Transactions on Robotics*, vol. 23, no. 4, pp. 693–703, 2007.

[18] B. J. Karaki and M. S. Mahmoud, "Scaled consensus for multiagent systems under denial-of-service attacks and exogenous disturbance," *International Journal of Systems Science*, vol. 53, no. 1, pp. 108–121, 2022.

[19] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[20] Y. Q. Chen and Z. Wang, "Formation control: a review and a new consideration," in *2005 IEEE/RSJ International conference on intelligent robots and systems*, pp. 3181–3186, IEEE, 2005.

[21] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on automatic control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[22] R. Horaud, "A short tutorial on graph laplacians, laplacian embedding, and spectral clustering," *URl: http://csustan. csustan. edu/˜ tom/Lecture-Notes/Clustering/GraphLaplacian-tutorial. pdf*, 2009.

[23] M. Mesbahi and M. Egerstedt, "Graph theoretic methods in multiagent networks," 2010.

[24] B. J. Karaki and M. S. Mahmoud, "Quantised scaled consensus of linear multiagent systems on faulty networks," *International Journal of Systems Science*, vol. 52, no. 8, pp. 1692–1706, 2021.

[25] Q. Song, F. Liu, H. Su, and A. V. Vasilakos, "Semi-global and global containment control of multi-agent systems with second-order dynamics and input saturation," *International Journal of Robust and Nonlinear Control*, vol. 26, no. 16, pp. 3460–3480, 2016.

[26] B. J. Karaki and M. S. Mahmoud, "Distributed event-triggered consensus protocols for discrete-time multiagent systems," *IMA Journal of Mathematical Control and Information*, vol. 38, no. 4, pp. 1046–1071, 2021.

[27] D. Scharf, F. Hadaegh, and S. Ploen, "A survey of space formation flying guidance and control (part i)," in *American Control Conference*, 2004.

[28] M. Martin, P. Klupar, S. Kilberg, and J. Winter, "Techsat 21 and revolutionizing space missions using microsatellites," 2001.

[29] B. J. Karaki and M. S. Mahmoud, "Scaled consensus design for multiagent systems under dos attacks and communication-delays," *Journal of the Franklin Institute*, vol. 358, no. 7, pp. 3901–3918, 2021.

[30] R. Merris, "Laplacian matrices of graphs: a survey," *Linear algebra and its applications*, vol. 197, pp. 143–176, 1994.

[31] T. Rothvoss, "Design and analysis of algorithms,"

[32] H. S. Hasan, M. Hussein, S. M. Saad, and M. A. M. Dzahir, "An overview of local positioning system: Technologies, techniques and applications," *International Journal of Engineering & Technology*, vol. 7, no. 3.25, pp. 1–5, 2018.

[33] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Communications surveys & tutorials*, vol. 11, no. 1, pp. 13–32, 2009.

[34] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil, "Landmarc: Indoor location sensing using active rfid," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003).*, pp. 407–415, IEEE, 2003.

[35] K. Al Nuaimi and H. Kamel, "A survey of indoor positioning systems and algorithms," in *2011 international conference on innovations in information technology*, pp. 185–190, IEEE, 2011.

[36] P. Tomé, C. Robert, R. Merz, C. Botteron, A. Blatter, and P.-A. Farine, "Uwb-based local positioning system: From a small-scale experimental platform to a large-scale deployable system," in *2010 International Conference on Indoor Positioning and Indoor Navigation*, pp. 1–10, IEEE, 2010.

[37] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198, IEEE, 2016.

[38] C.-H. Chu, D.-N. Yang, and M.-S. Chen, "Image stablization for 2d barcode in handheld devices," in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 697–706, 2007.

[39] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE international conference on robotics and automation*, pp. 3400–3407, IEEE, 2011.

[40] S. Tiwari, "An introduction to qr code technology," in *2016 international conference on information technology (ICIT)*, pp. 39–44, IEEE, 2016.

[41] D.-H. Shin, J. Jung, and B.-H. Chang, "The psychology behind qr codes: User experience perspective," *Computers in Human Behavior*, vol. 28, no. 4, pp. 1417–1426, 2012.

[42] T. Tocci, L. Capponi, and G. Rossi, "Aruco marker-based displacement measurement technique: uncertainty analysis," *Engineering Research Express*, vol. 3, no. 3, p. 035032, 2021.

[43] A. S. Ismailov, Z. B. JoâĂŸRayev, *et al.*, "Study of arduino microcontroller board," *Science and Education*, vol. 3, no. 3, pp. 172–179, 2022.

[44] Y. Li, S. Dai, L. Zhao, X. Yan, and Y. Shi, "Topological design methods for mecanum wheel configurations of an omnidirectional mobile robot," *Symmetry*, vol. 11, no. 10, p. 1268, 2019.

[45] A. Gfrerrer, "Geometry and kinematics of the mecanum wheel," *Computer Aided Geometric Design*, vol. 25, no. 9, pp. 784–791, 2008.

[46] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th international conference on industrial and information systems (ICIIS)*, pp. 1–5, IEEE, 2012.

Appendices

.1 Datasheet

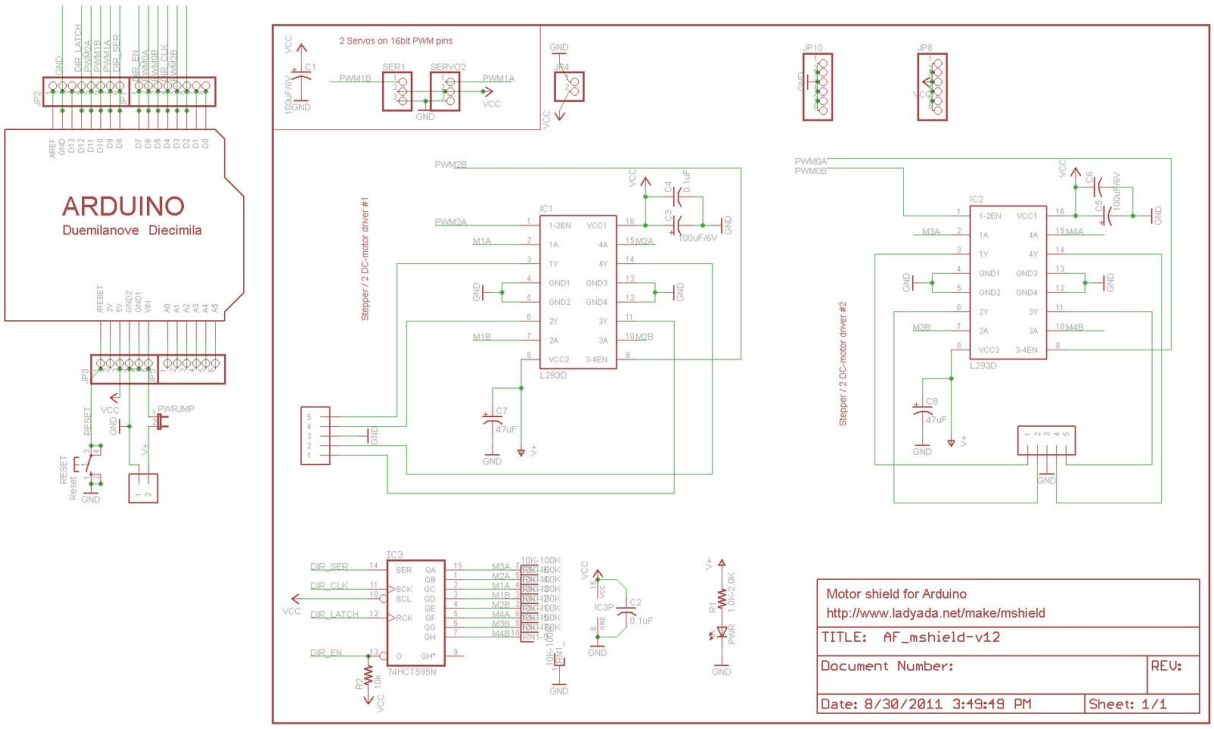


Figure 1: Motor Driver Shield L293D Datasheet

.2 Arduino Code Transmitter

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(10, 9);  // CE, CSN (Adjust pins as per your setup)
const byte address[6] = "00001";  // Must match receiver
int8_t received_data[24];  // Buffer for incoming serial data

void setup() {
    Serial.begin(57600);
    Serial.println("Arduino Nano Ready");
pinMode(5,OUTPUT);

    // NRF24 Setup
    radio.begin();
    radio.openWritingPipe(address);
    radio.setRetries(15, 0);  // 3 retries, 0ms delay (instead of default 15 retries, 4ms delay)
    radio.setPALevel(RF24_PA_HIGH);
    radio.setDataRate(RF24_1MBPS);
    radio.stopListening();  // Set as transmitter
}

// Function to read a valid 24-byte packet from Serial
bool read_serial_data() {
    while (Serial.available() >= 26) {  // Ensure a full packet (1 start + 24 data + 1 end)
        if (Serial.read() == 0xAA) {  // Look for Start Byte
            Serial.readBytes((char*)received_data, 24);  // Read 24 data bytes
            if (Serial.read() == 0x55) {  // Look for End Byte
                  return true;  // Successfully read a valid packet
            }
        }
    }

     return false;  // No valid packet found

}

void loop() {
     if( read_serial_data())
       {  // Only send if valid data was read
        bool success = radio.write(&received_data, sizeof(received_data));

    }

}
```

.3 Arduino Code Receiver

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

```

```
// RF24 Setup
RF24 radio(A5, A4);  // CE, CSN (use the correct pins for your setup)
const byte address[6] = "00001";  // Must match transmitter
int8_t received_data[24];  // Buffer for received data

// 74HC595 Shift Register Pins
#define CLOCK_PIN 4  // SHCP (Clock)
#define LATCH_PIN 2  // STCP (Latch)
#define SERIAL_PIN 8 // DS (Serial Input)
#define ENABLE_PIN 7 // Enable direction control

// Individual PWM speed control pins
#define M1_PWM 3  // Motor 1 Speed
#define M2_PWM 5  // Motor 2 Speed
#define M3_PWM 6  // Motor 3 Speed
#define M4_PWM 10 // Motor 4 Speed

// Variable to store the current state of the shift register
byte motorState = 0b00000000;

// Function to update shift register with new direction values
void updateShiftRegister() {
  digitalWrite(LATCH_PIN, LOW);
  shiftOut(SERIAL_PIN, CLOCK_PIN, MSBFIRST, motorState);
  digitalWrite(LATCH_PIN, HIGH);
}

// Function to set motor direction: 1 (Forward), -1 (Backward), 0 (Stop)
void setMotorSpeed(int motor, int speed) {
  switch (motor) {
    case 1: // Motor 1 (Bit 7 and Bit 1)
      analogWrite(M3_PWM, abs(speed));
      if (speed > 0)
        motorState = (motorState & 0b01011111) | 0b10000000; // Bit 7 HIGH, Bit 1 LOW (Forward)
      else if (speed < 0)
        motorState = (motorState & 0b01011111) | 0b00100000; // Bit 7 LOW, Bit 1 HIGH (Backward)
      else
        motorState &= (motorState & 0b01011111);  // Clear both bits (Stop)
      break;

    case 2: // Motor 2 (Bit 3 and Bit 6)
      analogWrite(M1_PWM, abs(speed));
      if (speed < 0)
        motorState = (motorState & 0b11101101) | 0b00010000; // Bit 6 HIGH, Bit 3 LOW (Forward)
      else if (speed > 0)
        motorState = (motorState & 0b11101101) | 0b00000010; // Bit 6 LOW, Bit 3 HIGH (Backward)
      else
        motorState &= (motorState & 0b11101101); // Clear both bits (Stop)
      break;

    case 3: // Motor 3 (Bit 5 and Bit 4)
      analogWrite(M4_PWM, abs(speed));
      if (speed > 0)
        motorState = (motorState & 0b11110011) | 0b00000100; // Bit 5 HIGH, Bit 4 LOW (Forward)
      else if (speed < 0)
        motorState = (motorState & 0b11110011) | 0b00001000; // Bit 5 LOW, Bit 4 HIGH (Backward)
      else
        motorState &= (motorState & 0b11110011); // Clear both bits (Stop)
```

```
      break;

    case 4: // Motor 4 (Bit 2 and Bit 0)
      analogWrite(M2_PWM, abs(speed));
      if (speed <0)
        motorState = (motorState & 0b10111110) | 0b00000001; // Bit 2 HIGH, Bit 0 LOW (Forward)
      else if (speed> 0)
        motorState = (motorState & 0b10111110) | 0b01000000; // Bit 2 LOW, Bit 0 HIGH (Backward)
      else
        motorState &= (motorState & 0b10111110); // Clear both bits (Stop)
      break;
  }
  updateShiftRegister(); // Apply the updated motor directions
}

void setup() {
  // Set shift register pins as output
      Serial.begin(9600);

  pinMode(CLOCK_PIN, OUTPUT);
  pinMode(LATCH_PIN, OUTPUT);
  pinMode(SERIAL_PIN, OUTPUT);
  pinMode(ENABLE_PIN, OUTPUT);

  // Set PWM pins as output
  pinMode(M1_PWM, OUTPUT);
  pinMode(M2_PWM, OUTPUT);
  pinMode(M3_PWM, OUTPUT);
  pinMode(M4_PWM, OUTPUT);

  // RF24 setup
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_LOW);
  radio.setDataRate(RF24_1MBPS);   // âÙĚ Change to 1 Mbps
  radio.startListening();  // Set as receiver
}

void loop() {


    while (!radio.available()) {
        // Loop blocks until new data arrives
    }

    radio.read(&received_data, sizeof(received_data));


    setMotorSpeed(1,  received_data[0]);
     setMotorSpeed(2,  received_data[1]);
    setMotorSpeed(3,  received_data[2]);
    setMotorSpeed(4,  received_data[3]);

   for (int i = 0; i < 4; i++) {
        Serial.print(received_data[i]);
        Serial.print(" ");
    }
    Serial.println();
```

```
}
```

.4 Python Code

```python
import apriltag
import cv2
import numpy as np
import time
import math
import serial
import struct
import threading

ser = serial.Serial('/dev/tty.usbserial-1430', 57600, timeout=0.1, write_timeout=0.015)

if not ser.is_open:
    ser.open()

data_lock = threading.Lock()
shared_data = [0, 0, -0, 0, 0, -0, 0, 0, -0, 0, 0, -0, 0, -0, 0, 0, -0, 0, 0, 0, -0, 0, -0, 0
april_tag_info = {}  # Shared AprilTag info


def send_data():
    time.sleep(0.1)

    while True:
        with data_lock:
            data_copy = [max(-128, min(127, x)) for x in shared_data]

        start_time = time.perf_counter()
        start_byte = 0xAA
        end_byte = 0x55
        packet = struct.pack('B24bB', start_byte, *data_copy[:24], end_byte)

        ser.write(packet)
        ser.flush()

        end_time = time.perf_counter()
        elapsed_time = (end_time - start_time) * 1000

        time.sleep(0.01)


def modify_data():
    class PIDController:
        def __init__(self, Kp, Ki, Kd, setpoint):
            self.Kp = Kp
```

```
            self.Ki = Ki
            self.Kd = Kd
            self.setpoint = setpoint
            self.previous_error = 0
            self.integral = 0

        def update(self, current_value, dt):
            error = self.setpoint - current_value
            self.integral += error * dt
           # if abs(error) < 50:  # reeset Control
               # if abs(error) > 5:
               #     self.integral = 20
            if abs(error) < 15:  # reeset Control
                self.integral = 0
            if abs(self.integral) > 120:  # reeset Control
                self.integral -= error * dt

            derivative = (error - self.previous_error) / dt if dt > 0 else 0
            output = self.Kp * error + self.Ki * self.integral + self.Kd * derivative
            self.previous_error = error
            return output

    class PIDControllerz:
        def __init__(selfz, Kp, Ki, Kd, setpoint=45):
            selfz.Kp = Kp
            selfz.Ki = Ki
            selfz.Kd = Kd
            selfz.setpoint = setpoint
            selfz.previous_error = 0
            selfz.integral = 0

        def update(selfz, current_value, dt):
            error = selfz.setpoint - current_value
            # error = (error + 180) % 360 - 180
            error = selfz.setpoint - current_value
            if error > 180:
                error = error - 360
            if error < -180:
                error = error + 360

            selfz.integral += error * dt
            if abs(error) < 3:  # reeset Control
                selfz.integral = 0
            derivative = (error - selfz.previous_error) / dt if dt > 0 else 0
            output = selfz.Kp * error + selfz.Ki * selfz.integral + selfz.Kd * derivative
            selfz.previous_error = error
            return output

    pid_x1 = PIDController(Kp=0.25, Ki=0.695, Kd=0.31, setpoint=500)
    pid_y1 = PIDController(Kp=0.25, Ki=0.695, Kd=0.31, setpoint=500)
    pid_z1 = PIDControllerz(Kp=0.628, Ki=0.162815, Kd=0.31, setpoint=45)
```

```
96
97      pid_x2 = PIDController(Kp=01.025, Ki=01.5, Kd=0.531, setpoint=500)
98      pid_y2 = PIDController(Kp=001.025, Ki=01.5, Kd=0.532, setpoint=500)
99      pid_z2 = PIDControllerz(Kp=01.0628, Ki=0.162815, Kd=0.12531, setpoint=45)
100
101     pid_x3 = PIDController(Kp=0.25, Ki=0.5, Kd=0.1, setpoint=500)
102     pid_y3 = PIDController(Kp=0.25, Ki=0.5, Kd=0.1, setpoint=500)
103     pid_z3 = PIDControllerz(Kp=0.628, Ki=0.62815, Kd=0.1, setpoint=45)
104
105     pid_x4 = PIDController(Kp=0.25, Ki=0.495, Kd=0.31, setpoint=500)
106     pid_y4 = PIDController(Kp=0.25, Ki=0.495, Kd=0.31, setpoint=500)
107     pid_z4 = PIDControllerz(Kp=0.629348, Ki=0.6294815, Kd=0.31, setpoint=45)
108
109     pid_x5 = PIDController(Kp=0.25, Ki=0.5, Kd=0.31, setpoint=500)
110     pid_y5 = PIDController(Kp=0.25, Ki=0.5, Kd=0.31, setpoint=500)
111     pid_z5 = PIDControllerz(Kp=0.625128, Ki=0.62815, Kd=0.5431, setpoint=45)
112
113     pid_x6 = PIDController(Kp=0.25, Ki=0.5, Kd=0.21, setpoint=500)
114     pid_y6 = PIDController(Kp=0.25, Ki=0.5, Kd=0.21, setpoint=500)
115     pid_z6 = PIDControllerz(Kp=0.628, Ki=0.862815, Kd=0.4, setpoint=45)
116
117     X = np.zeros((18, 1))
118     Vt = np.zeros((18, 1))
119     VttI = np.zeros((18, 1))
120
121     Wm = np.zeros((24, 1))
122
123     rx1, ry1, rz1 = 300, 300, 0
124     rx2, ry2, rz2 = 300, 900, 0
125     rx3, ry3, rz3 = 800, 300, 0
126     rx4, ry4, rz4 = 800, 900, 0
127     rx5, ry5, rz5 = 1300, 300, 0
128     rx6, ry6, rz6 = 1300, 900, 0
129
130     r = np.array([
131         rx1, ry1, rz1,
132         rx2, ry2, rz2,
133         rx3, ry3, rz3,
134         rx4, ry4, rz4,
135         rx5, ry5, rz5,
136         rx6, ry6, rz6
137     ])
138
139     X[0], X[1], X[2] = rx1, ry1, rz1
140     X[3], X[4], X[5] = rx2, ry2, rz2
141     X[6], X[7], X[8] = rx3, ry3, rz3
142     X[9], X[10], X[11] = rx4, ry4, rz4
143     X[12], X[13], X[14] = rx5, ry5, rz5
144     X[15], X[16], X[17] = rx6, ry6, rz6
145
146     x1, y1, angle1 = rx1, ry1, rz1
```

```
    x2, y2, angle2 = rx2, ry2, rz2
    x3, y3, angle3 = rx3, ry3, rz3
    x4, y4, angle4 = rx4, ry4, rz4
    x5, y5, angle5 = rx5, ry5, rz5
    x6, y6, angle6 = rx6, ry6, rz6


    L = np.array([
        [5, -1, -1, -1, -1, -1],
        [-1, 5, -1, -1, -1, -1],
        [-1, -1, 5, -1, -1, -1],
        [-1, -1, -1, 5, -1, -1],
        [-1, -1, -1, -1, 5, -1],
        [-1, -1, -1, -1, -1, 5]
    ])


    I1 = np.array([
        [1, 0, 0]
    ])
    I2 = np.array([
        [0, 1, 0]
    ])
    I3 = np.array([
        [0, 0, 1]
    ])

    while True:
        t = time.perf_counter()

        dt = 0.01  # Time step in seconds (same as sleep)

        with data_lock:
            for i in range(len(shared_data)):
                shared_data[i] = max(-128, min(127, int(Wm[i].item())))

            if 1 in april_tag_info:
                x1, y1 = april_tag_info[1]['center']
                angle1 = april_tag_info[1]['angle']
            else:
                x1=rx1
                y1=ry1
                angle1=rz1

            if 2 in april_tag_info:
                x2, y2 = april_tag_info[2]['center']
                angle2 = april_tag_info[2]['angle']
            else:
                x2 = rx2
                y2 = ry2
                angle2 = rz2
```

```
198
199
200             if 3 in april_tag_info:
201                 x3, y3 = april_tag_info[3]['center']
202                 angle3 = april_tag_info[3]['angle']
203             else:
204                 x3 = rx3
205                 y3 = ry3
206                 angle3 = rz3
207
208             if 4 in april_tag_info:
209                 x4, y4 = april_tag_info[4]['center']
210                 angle4 = april_tag_info[4]['angle']
211             else:
212                 x4 = rx4
213                 y4 = ry4
214                 angle4 = rz4
215
216             if 5 in april_tag_info:
217                 x5, y5 = april_tag_info[5]['center']
218                 angle5 = april_tag_info[5]['angle']
219             else:
220                 x5 = rx5
221                 y5 = ry5
222                 angle5 = rz5
223
224
225             if 6 in april_tag_info:
226                 x6, y6 = april_tag_info[6]['center']
227                 angle6 = april_tag_info[6]['angle']
228             else:
229                 x6 = rx6
230                 y6 = ry6
231                 angle6 = rz6
232
233         X[0, 0], X[1, 0], X[2, 0] = x1, y1, angle1
234         X[3, 0], X[4, 0], X[5, 0] = x2, y2, angle2
235         X[6, 0], X[7, 0], X[8, 0] = x3, y3, angle3
236         X[9, 0], X[10, 0], X[11, 0] = x4, y4, angle4
237         X[12, 0], X[13, 0], X[14, 0] = x5, y5, angle5
238         X[15, 0], X[16, 0], X[17, 0] = x6, y6, angle6
239
240         pid_x1.setpoint = np.kron(L[0, :], I1).flatten() @ r.flatten()
241         pid_y1.setpoint = np.kron(L[0, :], I2).flatten() @ r.flatten()
242         pid_z1.setpoint = np.kron(L[0, :], I3).flatten() @ r.flatten()
243
244         pid_x2.setpoint = np.kron(L[1, :], I1).flatten() @ r.flatten()
245         pid_y2.setpoint = np.kron(L[1, :], I2).flatten() @ r.flatten()
246         pid_z2.setpoint = np.kron(L[1, :], I3).flatten() @ r.flatten()
247
248         pid_x3.setpoint = np.kron(L[2, :], I1).flatten() @ r.flatten()
```

```
        pid_y3.setpoint = np.kron(L[2, :], I2).flatten() @ r.flatten()
        pid_z3.setpoint = np.kron(L[2, :], I3).flatten() @ r.flatten()

        pid_x4.setpoint = np.kron(L[3, :], I1).flatten() @ r.flatten()
        pid_y4.setpoint = np.kron(L[3, :], I2).flatten() @ r.flatten()
        pid_z4.setpoint = np.kron(L[3, :], I3).flatten() @ r.flatten()

        pid_x5.setpoint = np.kron(L[4, :], I1).flatten() @ r.flatten()
        pid_y5.setpoint = np.kron(L[4, :], I2).flatten() @ r.flatten()
        pid_z5.setpoint = np.kron(L[4, :], I3).flatten() @ r.flatten()

        pid_x6.setpoint = np.kron(L[5, :], I1).flatten() @ r.flatten()
        pid_y6.setpoint = np.kron(L[5, :], I2).flatten() @ r.flatten()
        pid_z6.setpoint = np.kron(L[5, :], I3).flatten() @ r.flatten()


        Lx1 = np.kron(L[0, :], I1).flatten() @ X.flatten()
        Ly1 = np.kron(L[0, :], I2).flatten() @ X.flatten()
        Lz1 = np.kron(L[0, :], I3).flatten() @ X.flatten()
        Lx2 = np.kron(L[1, :], I1).flatten() @ X.flatten()
        Ly2 = np.kron(L[1, :], I2).flatten() @ X.flatten()
        Lz2 = np.kron(L[1, :], I3).flatten() @ X.flatten()
        Lx3 = np.kron(L[2, :], I1).flatten() @ X.flatten()
        Ly3 = np.kron(L[2, :], I2).flatten() @ X.flatten()
        Lz3 = np.kron(L[2, :], I3).flatten() @ X.flatten()
        Lx4 = np.kron(L[3, :], I1).flatten() @ X.flatten()
        Ly4 = np.kron(L[3, :], I2).flatten() @ X.flatten()
        Lz4 = np.kron(L[3, :], I3).flatten() @ X.flatten()
        Lx5 = np.kron(L[4, :], I1).flatten() @ X.flatten()
        Ly5 = np.kron(L[4, :], I2).flatten() @ X.flatten()
        Lz5 = np.kron(L[4, :], I3).flatten() @ X.flatten()
        Lx6 = np.kron(L[5, :], I1).flatten() @ X.flatten()
        Ly6 = np.kron(L[5, :], I2).flatten() @ X.flatten()
        Lz6 = np.kron(L[5, :], I3).flatten() @ X.flatten()

        Vx1 = pid_x1.update(Lx1, dt)
        Vy1 = pid_y1.update(Ly1, dt)
        Vz1 = pid_z1.update(Lz1, dt)
        Vt[0] = -Vx1
        Vt[1] = Vy1
        Vt[2] = Vz1

        Vx2 = pid_x2.update(Lx2, dt)
        Vy2 = pid_y2.update(Ly2, dt)
        Vz2 = pid_z2.update(Lz2, dt)
        Vt[3] = -Vx2
        Vt[4] = Vy2
        Vt[5] = -Vz2

        Vx3 = pid_x3.update(Lx3, dt)
        Vy3 = pid_y3.update(Ly3, dt)
```

```
        Vz3 = pid_z3.update(Lz3, dt)
        Vt[6] = -Vx3
        Vt[7] = Vy3
        Vt[8] = Vz3

        Vx4 = pid_x4.update(Lx4, dt)
        Vy4 = pid_y4.update(Ly4, dt)
        Vz4 = pid_z4.update(Lz4, dt)
        Vt[9] = -Vx4
        Vt[10] = Vy4
        Vt[11] = -Vz4

        Vx5 = pid_x5.update(Lx5, dt)
        Vy5 = pid_y5.update(Ly5, dt)
        Vz5 = pid_z5.update(Lz5, dt)

        Vt[12] = -Vx5
        Vt[13] = Vy5
        Vt[14] = Vz5

        Vx6 = pid_x6.update(Lx6, dt)
        Vy6 = pid_y6.update(Ly6, dt)
        Vz6 = pid_z6.update(Lz6, dt)
        Vt[15] = -Vx6
        Vt[16] = Vy6
        Vt[17] = -Vz6

        theta1 = math.radians(angle1)
        theta2 = math.radians(angle2)
        theta3 = math.radians(angle3)
        theta4 = math.radians(angle4)
        theta5 = math.radians(angle5)
        theta6 = math.radians(angle6)

        R1 = np.array([
            [math.cos(theta1), -math.sin(theta1), 0],
            [math.sin(theta1), math.cos(theta1), 0],
            [0, 0, 1]
        ])
        R2 = np.array([
            [math.cos(theta2), -math.sin(theta2), 0],
            [math.sin(theta2), math.cos(theta2), 0],
            [0, 0, 1]
        ])
        R3 = np.array([
            [math.cos(theta3), -math.sin(theta3), 0],
            [math.sin(theta3), math.cos(theta3), 0],
            [0, 0, 1]
        ])
        R4 = np.array([
            [math.cos(theta4), -math.sin(theta4), 0],
```

```
            [math.sin(theta4), math.cos(theta4), 0],
            [0, 0, 1]
        ])
        R5 = np.array([
            [math.cos(theta5), -math.sin(theta5), 0],
            [math.sin(theta5), math.cos(theta5), 0],
            [0, 0, 1]
        ])
        R6 = np.array([
            [math.cos(theta6), -math.sin(theta6), 0],
            [math.sin(theta6), math.cos(theta6), 0],
            [0, 0, 1]
        ])

        VttI[0:3] = R1 @ Vt[0:3]
        VttI[3:6] = R2 @ Vt[3:6]
        VttI[6:9] = R3 @ Vt[6:9]
        VttI[9:12] = R4 @ Vt[9:12]
        VttI[12:15] = R5 @ Vt[12:15]
        VttI[15:18] = R6 @ Vt[15:18]

        MR = np.array([
            [1, 1, 1],
            [1, -1, 1],
            [-1, -1, 1],
            [-1, 1, 1]
        ])

        ML = np.array([
            [1, -1, 1],
            [-1, -1, 1],
            [-1, 1, 1],
            [1, 1, 1]
        ])

        Wm[0:4] = ML @ VttI[0:3]
        Wm[4:8] = MR @ VttI[3:6]
        Wm[8:12] = ML @ VttI[6:9]
        Wm[12:16] = MR @ VttI[9:12]
        Wm[16:20] = ML @ VttI[12:15]
        Wm[20:24] = MR @ VttI[15:18]

        time.sleep(dt)
        end_t = time.perf_counter()
        elapsed_t = (end_t - t) * 1000
        print(elapsed_t)


serial_thread = threading.Thread(target=send_data, daemon=True)
calculation_thread = threading.Thread(target=modify_data, daemon=True)

```

```
serial_thread.start()
calculation_thread.start()

options = apriltag.DetectorOptions(families="tag16h5")
detector = apriltag.Detector(options)

print("[INFO] Starting video stream...")
cap = cv2.VideoCapture(0)

cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1900)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)


tag_data_list = []

while True:

    start_time = time.time()

    ret, frame = cap.read()
    if not ret:
        print("[ERROR] Failed to capture image")
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    _, thresh = cv2.threshold(blurred, 150, 255, cv2.THRESH_BINARY)

    # print("[INFO] Detecting AprilTags...")
    results = detector.detect(thresh)
    # print(f"[INFO] {len(results)} total AprilTags detected")

    tag_data_list.clear()

    with data_lock:
        april_tag_info.clear()

        for result in results:
            (top_left, top_right, bottom_right, bottom_left) = result.corners
            top_left = (int(top_left[0]), int(top_left[1]))
            top_right = (int(top_right[0]), int(top_right[1]))
            bottom_right = (int(bottom_right[0]), int(bottom_right[1]))
            bottom_left = (int(bottom_left[0]), int(bottom_left[1]))

            center_x = int((top_left[0] + bottom_right[0]) / 2)
            center_y = int((top_left[1] + bottom_right[1]) / 2)
            center = (center_x, center_y)
            angle = math.degrees(math.atan2(top_right[1] - top_left[1], top_right[0] - top_le

            tag_data = {
                "id": result.tag_id,
```

```
                "center": center,
                "angle": angle
            }
            tag_data_list.append(tag_data)

            april_tag_info[result.tag_id] = {
                'center': center,
                'angle': angle
            }

            cv2.polylines(frame, [np.array([top_left, top_right, bottom_right, bottom_left],
                          isClosed=True, color=(0, 255, 0), thickness=2)
            cv2.circle(frame, center, 5, (255, 0, 0), -1)
            cv2.putText(frame, f"ID: {result.tag_id}", (top_left[0], top_left[1] - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
            cv2.putText(frame, f"({center_x}, {center_y})", (center_x + 10, center_y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
            cv2.putText(frame, f"Angle: {angle:.2f}Âř", (center_x + 10, center_y + 20),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

    cv2.imshow("AprilTag Detection", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    cycle_time = time.time() - start_time

cap.release()
cv2.destroyAllWindows()
```